May 2019

The Effectiveness of Codesters in Teaching Basic Computer Science Topics

Antonio Pedro Moctezuma
University of Wisconsin-Milwaukee

THE EFFECTIVENESS OF CODESTERS IN TEACHING BASIC COMPUTER SCIENCE TOPICS

by

Antonio Moctezuma

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin-Milwaukee

May 2019

ABSTRACT

THE EFFECTIVENESS OF CODESTERS IN TEACHING BASIC COMPUTER SCIENCE TOPICS

by

Antonio Moctezuma

The University of Wisconsin-Milwaukee, 2019

Under the Supervision of Professor Christine Cheng

Founded in 2014, Codesters is a visual programming environment (VPE) like the popular Scratch and Alice. Its goal is to teach middle school and older student's computer programming. Unlike its predecessors, users of Codesters drag and drop actual Python code instead of blocks and can edit the code themselves. Codesters has also developed modules that integrate coding lessons into the VPE.

In this study, we consider the Codesters Python 1 module and investigate its effectiveness in teaching the basic coding concepts of variables, loops and conditionals. During Fall 2018 and Spring 2019, we ran a coding class for eighth graders at a local Milwaukee school based on this module. We gave a pre-test, three quizzes and a post-test to evaluate what the students have learned. We then analyzed the results of these evaluations and compared them to those taken by students who learned programming in a traditional CS1 class.

Our results indicate that users of Codesters understood loops and conditionals as well as the students from the traditional CS1 class. We also found that the pre-test was a poor indicator of students' performance in the coding class suggesting that Codesters is able to engage students who might not necessarily excel in a traditional classroom.

To

My mom and dad for always believing in me in everything I do and allowing me to chase my dreams, I wouldn't be where I am today without you both,

My grandma and late grandpa for always having my back and supporting me throughout my entire college career,

My sister and her little family for being the best support system I could've ever asked for,

Desire for giving me the motivation to keep going no matter what, so I could be the best role model you could've ever asked for, you'll do great kid,

Shaina for always supporting me in the toughest times and for putting up with me even when you were equally as stressed,

Dr. Cheng, for assisting with every aspect with this thesis and for not giving up on me when the going got tough,

And Tyler for always being there when I needed a break from school whether it was at one of our places or in a booth at Pepi's.

I love you all!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

We would first like to thank Codesters and Mr. Gordon Smith for allowing us to use Codesters' name and cooperating with this research of their platform. Without your contributions we wouldn't have received your very important perspective on coding in K-12. We look forward to continually working with you on future projects and are excited to see what new modules you bring to the computer science education scene.

Thanks to Robert Sorenson, senior lecturer at the University of Wisconsin – Milwaukee, for allowing us to utilize your quiz questions in our research. Without your contribution we wouldn't have had a benchmark to compare our experimental group against. We also would like to thank you for always going above and beyond the call of duty in teaching your introductory programming course and inspiring us to study those beginning topics.

Thank you to the parents who gave permission for their children's scores to be included in this study. You have contributed to the beginning of something that will only get better as time goes on so your contribution will not go unnoticed. Thanks also to the UWM students who volunteered their scores as a part of our research. It has been an absolute joy to watch all of you grow as computer scientists but more importantly thank you for helping advance computer science education.

Thank you to Scratch and Alice for allowing us to use information about their platforms in our work. We also look forward to utilizing these other VPEs for future research field. Thank you for everything you do!

Section 1: Introduction

With new technological advances changing everyday life as we know it, it is important for everyone to realize the importance of computer programming and making sure the future generation has this skill. Many countries have adopted an initiative to integrate computer programming into the pre-college curriculum as early as elementary school to some mixed results and opinions [HA17].

In the US, some states like Wisconsin have laid out goals and expectations for Computer Science education at the K-12 level [PD17]. They want students to gain skills specific to the following five strands of computer knowledge: algorithms and programming, computing systems, data and analysis, impacts of computing, and networks and the internet.

The Wisconsin Department of Public Instruction (DPI) has created benchmarks for students at the grade levels of K-2, 3-5, 6-8, and 9-12. [DPI] For example, DPI expects students who have completed the grade 3-5 level to have the ability to "construct and execute algorithms, which include sequencing, loops, and conditionals to accomplish a task, both independently and collaboratively, with or without a computing device." These standards are quite new and is expected to prepare the next generation for future jobs.

At the other end of the spectrum, AP Computer Science A classes teach Java and cover essentially the same topics as an introductory college programming course (i.e., a CS 1 class). For example, students learn how to utilize variables in Java as well as write loops and conditionals to achieve some goal [CB14]. Of course the there are other more challenging material as well.

Younger students, however, cannot be taught Java. They need programming languages and environments that are less intimidating, more intuitive and easy to manipulate. Visual Programming environments, or VPEs, have emerged to address this need. Unlike traditional programming, VPEs such as Scratch and Alice do not show users the actual code. Instead, there is a panel on the screen that contains "blocks" representing small segments of code. Users drag and drop blocks to the screen and connect them together to create a program. This kind of "block programming" eliminates typos (a source of great frustration for many beginner programmers) and allows users to focus more on concepts rather than syntax. Users of VPEs are most likely to create programs that convey a story -- sprites (figures) are moving on the screen and the scenery is changing. It is also easy to make the programs interactive and to share the programs with other users.

The ease of using VPEs allow students to explore the available blocks and sprites to create their own projects. The figures and scenery are bright and colorful so they easily grab the students' attention. There are currently more than 20 different educational VPEs, but Scratch is the leader in this field. It is marketed as a tool for all ages where any K-12 student can use Scratch to make their own project.

VPEs are tools that allow users to explore coding. Without proper instruction, however, it is difficult for students to learn the important concepts of coding by just using a VPE. Thus, K-12 instructors will have to build a coding curriculum and then use the VPE as the programming language much like how an AP Computer Science teaches a class using Java. Recently, however, there is a new VPE that has created modules that integrate the curriculum into the VPE. It is called Codesters [C14].

Started in 2014, Codesters is a VPE like Scratch in that it provides a visually rich environment for coding. It is aimed at middle school students and combines the good aspects of a VPE with a traditional programming language. Codesters reach extends far past its home state of New York, however. They have a relatively large user base and have recently announced that they will be in every Tier 1 New York school [C14]. Codesters' goal is not only to teach students how to code but how to apply that code to their everyday school work. It also has features that makes it a very attractive platform for teaching coding.

First, it has several modules and its Python 1 module in particular has a built in curriculum that covers the basic coding concepts covered in a traditional beginning programming class (CS1) at the college level. The module begins with variables, moves on to loops, has some lessons on conditionals, a lesson on lists, and ends with incorporating all topics together. Second, like the other VPEs, the platform uses the drag-and-drop system to create code; but unlike them, what is dragged-and-dropped to the screen is actual Python code and not blocks. This has the important benefit of showing students what actual code looks like. It also provides the students the option of typing code, something that older students seem to prefer. Founder and CEO of Codesters Gordon Smith said that "we wanted to make it seem that the students were doing something real and not just sticking blocks together."

An instructor who knows how to code can easily use the Codesters Python 1 module as the backbone of a coding class at the late elementary, middle or early high school level. Indeed, the UW-Milwaukee Girls Who Code program has utilized it for its Level 1 class since Spring 2016. The coding classes that are run at various Milwaukee-area schools and

funded by the NSF grant *CS10K: Priming the PUMP - Preparing Urban Milwaukee for Principles of Computer Science* have also been using the module.

But is the Codesters Python 1 module effective in teaching the basics of coding? Is it able to engage students across a wide range of academic abilities? Do students gain a good understanding of the various coding concepts? To help us answer these questions, we ran a coding class for eighth graders at the Central City Cyber School in inner city Milwaukee. We supplemented the Codesters Python 1 module with some lectures but it was otherwise the main way for students to learn how to code. We gave them a pre-test, quizzes and then a post-test to evaluate what they learned. We then compared the results of their tests with those of college students who took CS 250, a beginning programming class, at UW-Milwaukee.

Section 2 discusses some studies related to our work. Section 3 compares Codesters with Scratch and Alice. Section 4 provides the background information and the logistical details of the study. Section 5 presents the results. Section 6 contains the conclusion and future work.

Section 2: Related Work

In this section, we describe prior work that others have done on the effectiveness of Visual Programming Environments to teach basic programming concepts. We consider three studies in particular because they relate to our current work the best. All of them make use of Scratch. As far as we know, no studies have been done on Codesters.

Section 2.1: From Scratch to "Real" Programming

The first study of interest to us [AMB15] are from three researchers of the Weizmann Institute of Science in Rehovot, Israel. Michal Armoni, Orni Meerbaum-Salant and Mordechai Ben-Ari aimed to answer the question -- "To what extent can middle school students learn Computer Science (CS) and how does it affect their encounter with CS in secondary school?" This team theorized that learning Scratch at the middle school level would "facilitate learning CS at the secondary school level both in terms of cognitive aspects and in terms of affective aspects."

The study took place during the 2011-12 school year. All participants were in 10th grade, 15 or 16 years old, and were enrolled in their first CS course in some Israeli high school. There were a total of 120 participants spanning five classes in four different high schools. Of these participants, 44 had previously taken a CS class in middle school and represented the experimental group of the study. The other 76 participants had no prior exposure to CS concepts and represented the control group of the study. These two groups

were mixed together in their respective schools. The high school CS classes based on an established curriculum taught the following concepts: variables, conditional execution, bounded repeated execution and conditional repeated execution.

The researchers used a mixed-method approach in the evaluation portion of the study. Students were given tests to assess how much they learned. In total, there were eight tests -- six of them concentrated on actual concepts, a midterm and then a final test. The researchers also observed the students in the classroom and conducted interviews. Once all the quantitative and qualitative data were gathered, the research team analyzed the results for each concept using a two-dimensional taxonomy that utilized Bloom's and the SOLO taxonomies.

The results were not as clear cut. For the concepts of variables and conditional execution, the scores of the experimental and control groups were in the same range. The study concluded that there was no significant difference between the datasets of the two groups even though the students in the experimental group had prior exposure to the said concepts. For the concepts of bounded and conditional repeated execution (i.e., loops), however, a significant difference between the two groups was found at all cognitive levels. The experimental group performed better than the control group. On the final test, although the experimental group scored better than the control group, the only part where they significantly differed were on the questions testing students' abilities to write code for a particular task. The research team concluded that although the results of the study may seem negative, the fact that Scratch was able to help with the concept of loops was a step in the right direction

especially since college students learning programming for the first time struggle with loops as well.

Section 2.2: Comparing Loop Misconceptions in block and text based programming languages at the K-12 Level

This second study [MBZ18] was conducted by Monika Mladenovic, Ivica Boljat and Zana Zanko of the University of Split in Croatia. They sought to analyze the students' misconceptions about loops at the K-12 level as opposed to the usual population of beginning undergraduate Computer Science students. The researchers believe that if students were taught basic programming concepts properly at a younger age, they will learn programming better at the high school or undergraduate level.

The study had 207 students from three elementary schools. The researchers used Scratch, Logo and Python to teach basic programming concepts. Scratch is a virtual programming language/environment where users can drag and drop "blocks" to create code. Logo is an educational programming language that makes use of turtle graphics and text. Python is a textual programming language and is seen by many as a good language for beginners.

The researchers conducted a pre-test to evaluate students' problem solving abilities. The students were then divided into three groups based on their scores. Of the 207 students, 73 were placed in the strong group, 63 in the intermediate group and 71 in the weak group. Students within each group were then randomly assigned to a class that taught programming using Scratch, Logo or Python. The classes spent three weeks teaching loops. A

post-test was conducted at the end to test the students' understanding of loop concepts. The test had five multiple choice questions.

The results of the study show that students with intermediate to low problem solving skills were more successful in Scratch than Logo or Python. Furthermore, based on the results of the post-test, students who learned loops using Scratch were less prone to loop misconceptions than students who used Logo or Python. Thus, there's an argument to be made that Scratch does facilitate the learning of loops at the elementary-age level.

Section 2.3. Improving Programming Skills in Engineering Education through Problem-Based Game Projects with Scratch

Scratch's intended audience are students in the K-8 age group but this has not stopped Computer Science educators from using Scratch with older students. In this third study [TC18] by Damla Topalli and Nergiz Ercil Cagiltay of Atilim University in Turkey, they wanted to improve the skills gained by the undergraduate students in the programming classes at their university. The researchers attributed a lack of math skills, overconcentration on syntax as opposed to concepts and the inability to see the large picture as issues that plagued many of their students. Their solution was to introduce an "enriched" beginner programming class (a CS1 course) where Scratch is incorporated into the curriculum. They then compared the performance of the students in this class as well as their Senior Project course against the students who took a traditional CS1 course at their university.

The curricula of the enriched CS1 course and the traditional CS1 course were identical except in two areas: in the enriched CS1 course, students used Scratch in the last 15 minutes of

their lab activity in lieu of the programming language the class was using; at the end of the semester, the students also had to submit a Scratch game project that taught Science or English and the project accounted for 15% of their grade. The researchers sought to answer these questions: did students in the enriched CS1 course receive a better grade than students in the traditional CS1 course? How about in their Senior Project? How about in their engineering program as a whole?

Using an independent sample t-test, the study found that students in the enriched CS1 course did perform significantly better than those in the traditional CS1 course. Another sample t-test showed that students who took the enriched CS1 course also performed better in their Senior Project course compared to the students in the traditional CS1 course. Finally, the study found that there was no significant difference between the two groups in their overall performance as engineering majors.

Section 3: Comparison of VPEs

The focus of this study is the visual programming environment (VPE) Codesters. It is not as well-known as other VPEs such as Scratch and Alice 3. In this section, we will compare the three VPEs using the first lesson in Codesters' Python 1 module, Dance Steps.

Section 3.1: Codesters

Let us begin with an overview of the Codesters environment below:

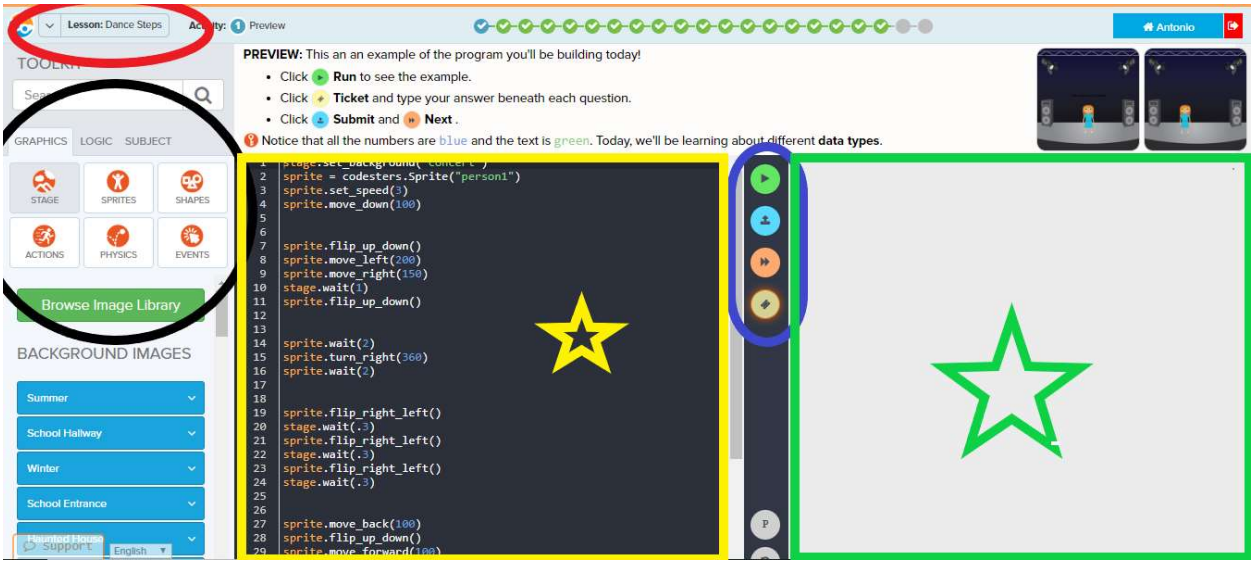

Figure 1: Codesters

We have marked important buttons and areas of the coding environment with colored boxes and shapes. The *code editor* is surrounded by a yellow square with a yellow star. It is where the user directly types their Python code or drag-and-drops code snippets from the blue tabs on the lower left corner of the editor. The *output area* is surrounded by a green square with a green star. It displays the output of the user's code once they hit the *green play button*. The other buttons inside the blue oval are the *blue submit button*, the *orange next*

button and the *yellow ticket button*. The user presses the blue button to submit their work, the orange button to move on to the next instruction and the yellow button to answer short questions related to the current lesson.

The *lesson selection box* is in the upper left corner and marked with a red oval. It displays the current lesson; if clicked, the user can change to a different lesson. *Categories of code snippets* are located on the left side and marked with a black oval. The three main categories are Graphics, Logic and Subject. Each category consists of subcategories which in turn have varying options as well. For example, under the Action subcategory of Graphics, a user can choose different actions that a sprite can perform.

The Codesters Python 1 module consists of 21 lessons. Each lesson has roughly 18 to 25 steps and focuses on a particular coding concept. At each step, the user must add a few lines of code, edit the current code, run the code, answer questions in the ticket or debug a given code snippet. The user has completed the lesson when all the steps are completed. The progress bar for a lesson is located on top of the code editor. If the user has not attempted a step yet, the corresponding circle is gray. If the user completed the step properly, a green check mark appears; otherwise, a red exclamation point appears.

We consider the first Codesters Python 1 Lesson, *Dance Steps*, and use it to compare Codesters to Scratch and Alice. Below are the 21 steps that a user must complete in the *Dance Steps* lesson.

1. Run the given code and answer some questions about the results

2. Set the background for the dance scene and add in a sprite. (Codesters suggests using a concert background and to add person4)

3. Change the sprite in the scene

4. Make the sprite move down to create the first dance step (Codesters suggests the user move their sprite down 100 pixels)

5. Look at the given code and answer questions about the string and integer variables used in the code

6. Debug the code so the code runs successfully. Buggy code is given to the user that they must fix.

7. Another debugging question much like step 6.

8. Flip the sprite so it appears to be standing on its head

9. Have the sprite move across the stage on its head

10. Move the sprite back to the other side of the screen

11. Flip the sprite so it is standing back up

12. Debug the code

13. Change the movement speed of the sprite (Codesters suggests using 3)

14. Debug the code

15. Have the sprite appear to flip by having them use the turn left command

16. Have the sprite face towards the left and add a wait command to the bottom of the code

17. Add 6 more copies of the two commands added in 16

18. Debug some lines of code

19. Run the code and answer questions about it

20. Customize the code given by changing the background or sprite, adding more dance commands, and adding another sprite

21. Make a new dance scene from scratch

Figure 2 below shows the code that accomplishes all movement actions required for *Dance Lesson* in Codesters. Steps 1, 5-7, 12-14 and 18-19 cannot be replicated in Scratch nor Alice because they are evaluating the user's understanding of a concept or code. Such evaluations are not part of the two VPEs.

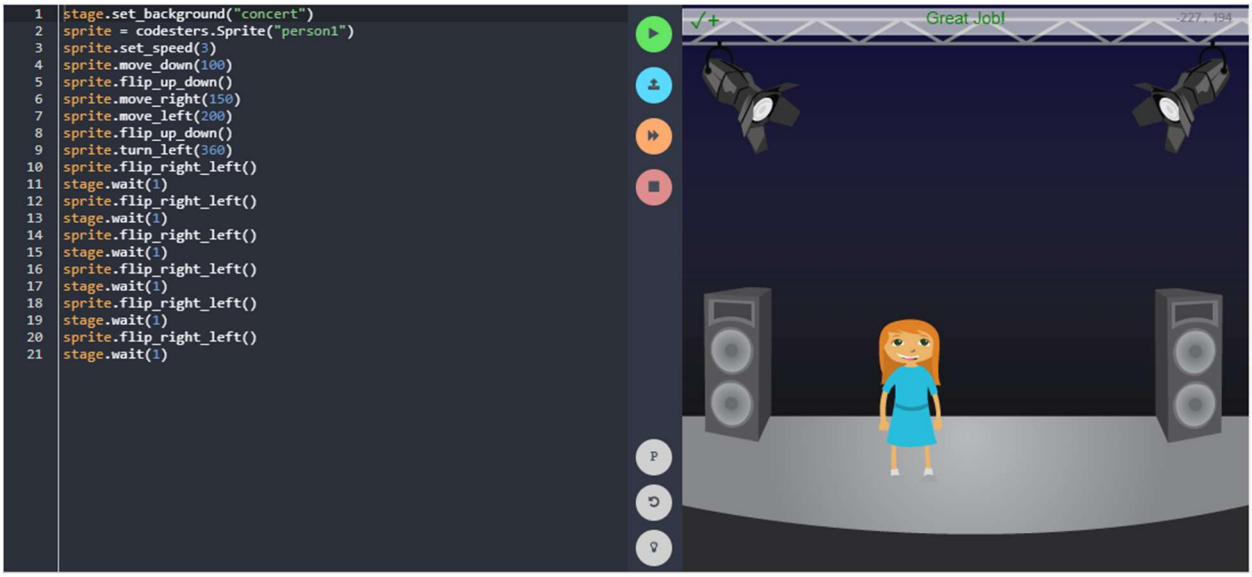

Figure 2: Code Snippet from Codesters

Section 3.2: Scratch

Scratch [S07] was created by the Lifelong Kindergarten group of the MIT Media Laboratory and is currently one of the most well-known VPEs. Scratch encourages students to

use their imagination, explore the Scratch programming environment, discover the various things they can code, and finally share it with the world through their public forum. Scratch's slogan is one that is simple yet to the point "Imagine, Program, Share." Like Section 3.1, we describe the layout and components of Scratch's workspace as shown below and how the user navigates this space.

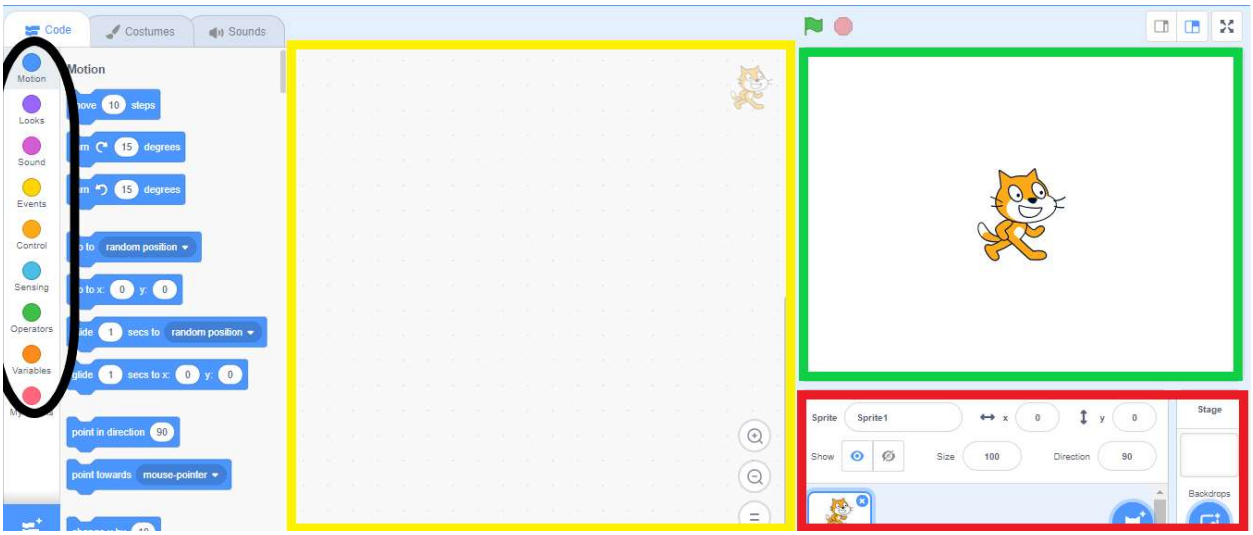

Figure 3: Scratch

We used the same colored boxes and shapes as the Codesters diagram to indicate the different parts of the Scratch VPE. The code editor is surrounded by a yellow square. Blocks of different colors and shapes are dragged from the left side and dropped into the editor. Each block looks like a puzzle piece and represent a snippet of code. Blocks are connected together so Scratch recognizes the whole block as a segment of code.

Just like in Codesters, the blocks are split into groups based on the code snippets they represent. The categories are Motion, Looks, Sound, Events, Control, Sensing, Operators, Variables, and My Blocks. Motion has the blocks which allow the sprite to move while Control contains blocks associated with loops and conditionals, etc.

Once blocks are placed in the code editor, the user must click the connected blocks in order to run the code. The result is displayed in the output area, which is surrounded by a green rectangle.

When a user first begins a project, the default sprite is the Scratch mascot, the Scratch Cat, and the default background is blank. The user can modify both by using the tabs within the red rectangle. There are a wide variety of sprites and backgrounds to choose from. Users can also create and upload their own to use in their projects. Note that a sprite in Scratch is a single image. Thus, if a sprite is facing a certain direction and asked to move a certain way, it will just move horizontally or vertically with no change to the image. If the user were to make the sprite "turn to face the other direction", that sprite would appear upside down because there is no way to alter the image. This will be illustrated in the output of the actions performed to emulate Dance Steps.

The left side of Figure 4 displays the sequence of blocks in Scratch that correspond to steps 2-4, 8-11, 15-17 in the Dance Step lesson of Codesters. The output of the code is on the right side. We modified the sprite and background so they were as similar to Codesters as possible. We also note that Scratch does not allow the user to control the speed of a sprite's movements, but Codesters does. If they want to display each dance step clearly, they have to insert wait commands between each move. Since movements in Codesters are a bit more fluid, a user can see every movement as opposed to Scratch which is more abrupt.

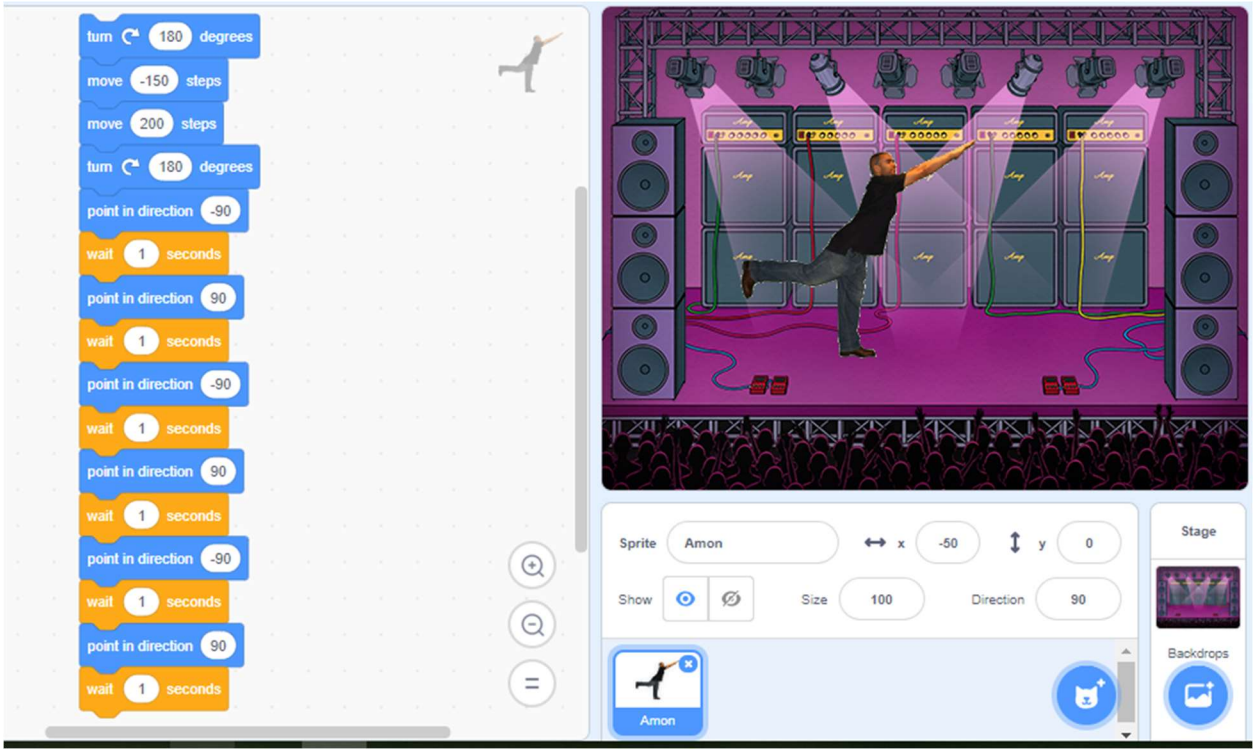

Figure 4: Scratch Code Snippet

Section 3.3: Alice

The last VPE we consider is Alice 3 [A95], created by the late Randy Pausch. The goal of Alice is to teach users how to code by having them create scenes to tell a story. There are several differences between Alice and Codesters. First, Alice is a 3D tool. In Codesters and Scratch, the user specifies the location of their sprite using the x and y coordinates. In Alice, they have to add a z coordinate. Second, Alice like Scratch, does not have a built in curriculum to guide its users through the learning process and instead relies on the exploration of the user. To quote Alice's website: "Unlike many of the puzzle-based applications Alice motivates through creative exploration. Alice is designed to teach logical and computational thinking skills, fundamental principles of programming and to be the first exposure to object-oriented

programming." Third, Alice utilizes object-oriented programming (OOP) concepts. When a user looks at their coding portion of Alice, there are many usages of these OOP concepts, which will more than likely seem foreign to them. However, the user is still dragging and dropping blocks like Scratch. Lastly, Alice is not web-accessible. It must be downloaded and used offline.

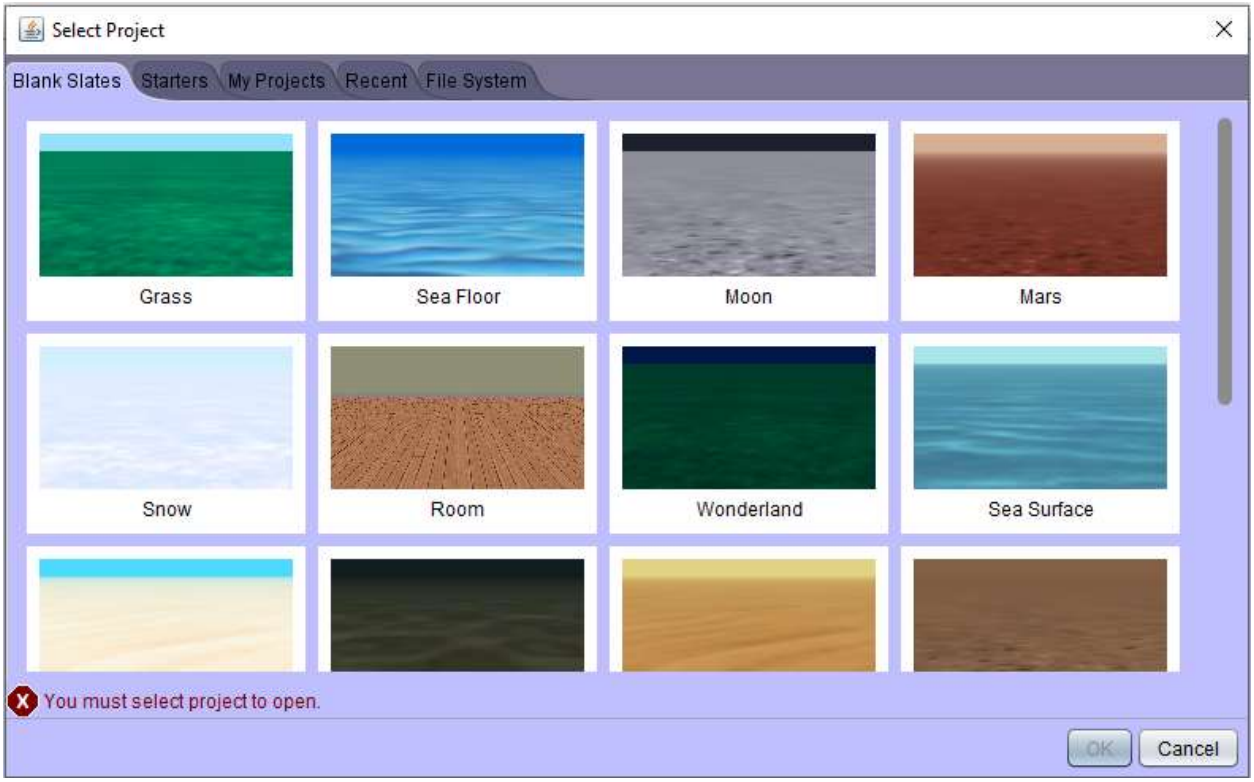

Figure 5: Alice New Project Screen

When Alice is launched, the user is presented with the screen shown in Figure 5. The user has to decide what background to use before they can work on the rest of the project.

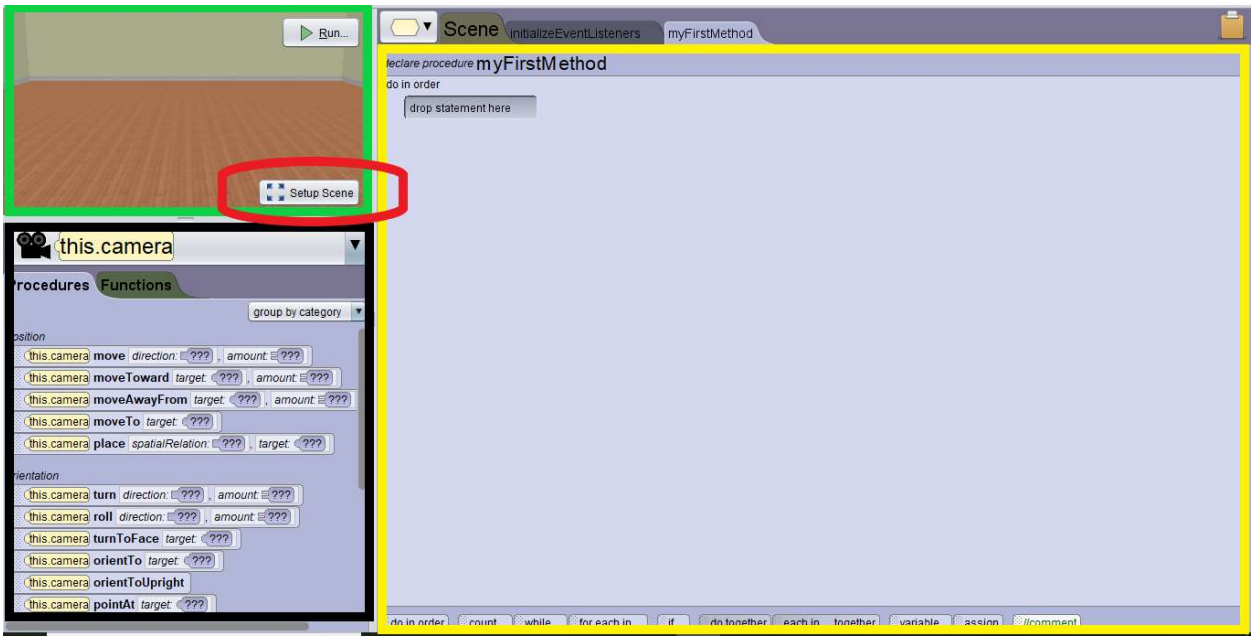

Figure 6: Alice Beginning Workspace

Once a background is selected, the user is presented with the image in Figure 6. The first button to note is the Setup Scene, marked with a red oval. It switches between the two views in Alice, the *code view*, which displays the code (as shown in Figure 6), and the *scene view,* which allows the user to modify or add objects to the scene of their story (as shown in Figure 7). In the code view, the code editor is surrounded by a yellow rectangle. The user drags and drops their code from the black box on the left to the code editor. Currently, the yellow square is showing the 'main' part of this program. Aside from the 'main' program, methods can be created and called from 'main' which is why there is a myFirstMethod tab. Finally, the green box shows what the scene currently looks like. The play button in the box runs the code the user has created.

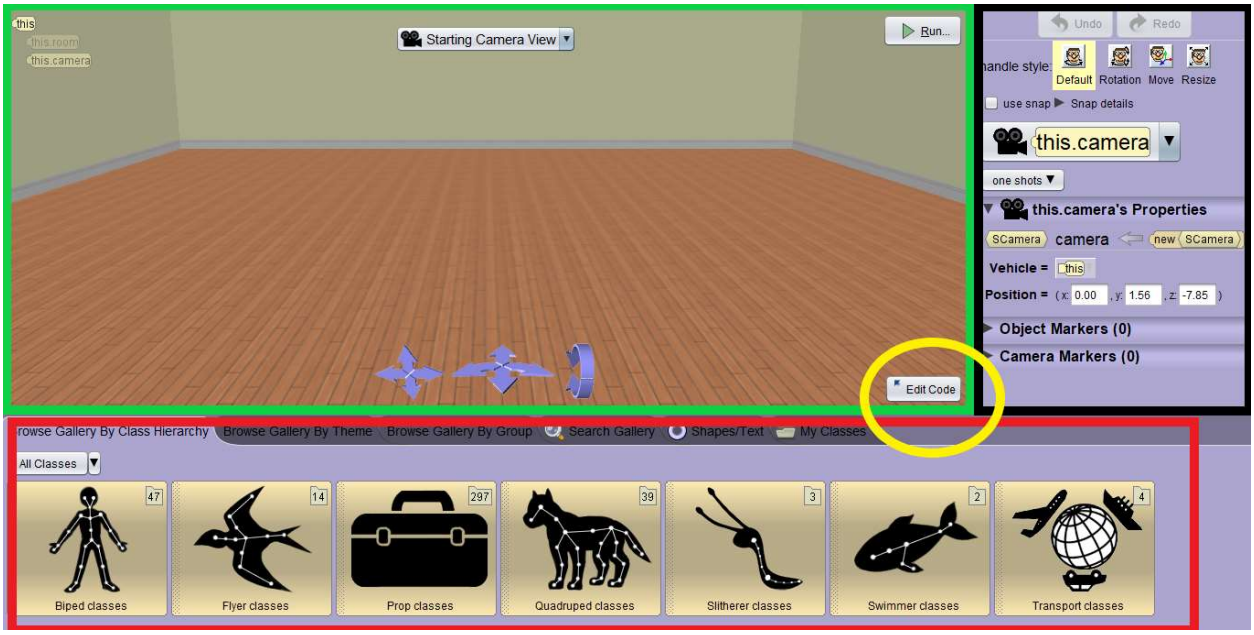

Figure 7: Alice Room View

Figure 7 displays the scene view of Alice which allows users to drag and drop sprites or other props into the scene. To switch back to the code view, the user simply clicks on the Edit Code button which has been identified with the yellow oval. The scene displayed in the code view is now enlarged and made interactive in the scene view. Users can drag items from the red rectangle and drop them anywhere inside the scene. When an object is selected, its properties such as position and color are displayed on the right in the black box.

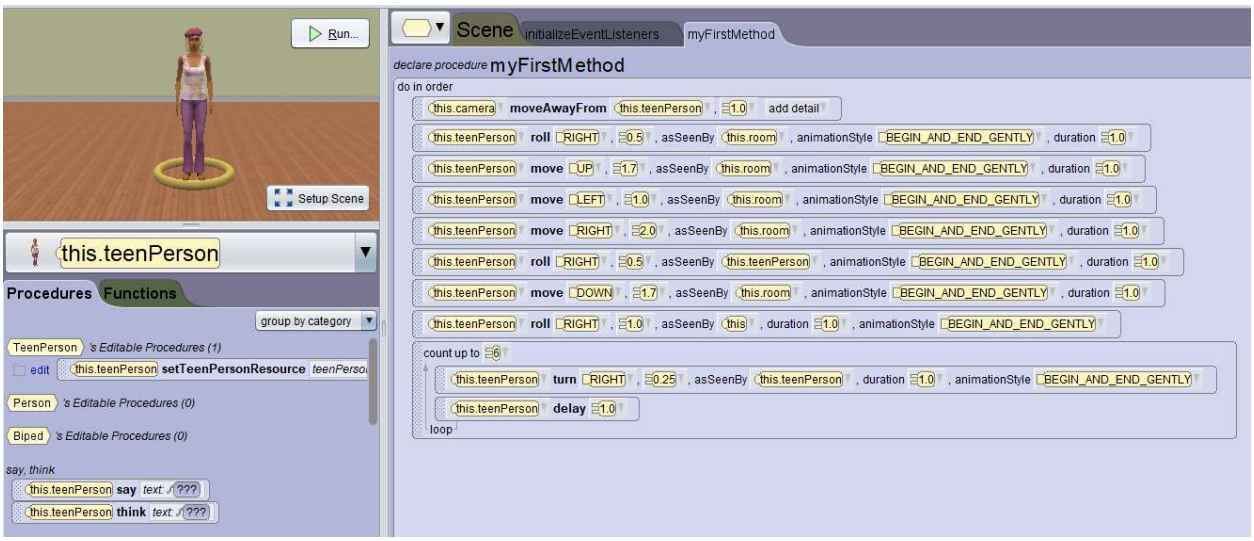

Figure 8: Alice Code Snippet

In Figure 8, we created the code corresponding to the requirements of the Dance Steps lesson. Again, we did not do steps 1, 5-7, 12-14, and 18-21. In order to fit more in the picture, we opted to use a loop instead of repeating 6 lines of code as was required by step 17. There were a few issues doing Dance Steps in Alice because of its 3D setup. For instance, it was difficult to have a sprite move down as this would cause it to go through the floor and not be visible to the user. Additionally, Alice did not have a way to change the speed of the sprites actions but it did have the ability to change the animation speed in between actions. This is done using blocks in the animationStyle area of each of these movement blocks. The user could make it so each movement begins and ends abruptly or gently.

What really separates Codesters Python 1 module from Scratch and Alice is that it has a built-in curriculum and that it shows users real code as opposed to blocks. This allows Codesters to teach both the ideas behind programming and the syntax of Python. Although Scratch and Alice can accomplish similar goals, they lack the capability of guiding a user through the coding

process. They instead promote discovery and exploration. For these reasons, we were interested in researching the effectiveness of Codesters.

Section 4: Experiment and Background Information

For our study on the Codesters Python 1 module, we taught a coding class for 15 weeks -- 10 weeks during Fall 2018 and 5 weeks during Spring 2019 -- at Central City Cyber School, a school located in Milwaukee's inner city. The students were all 8th graders from Mr. David Greenheck's homeroom. The coding class ran twice a week on Tuesdays and Thursdays from 10:30 - 11:30 AM in place of a regular study hall. The class was funded through the NSF grant *CS10K: Priming the PUMP - Preparing Urban Milwaukee for Principles of Computer Science*.

During the first week of class, students were given a pre-test to evaluate their math and reasoning skills. We wanted to see if students who performed well on this test will also do well with the Codesters lessons. During subsequent meetings, a typical session proceeded as follows: we introduced a concept, presented some examples to reinforce the concept and then had students do actual coding on the related Codesters lesson after the instruction. We then gave short quizzes after the topics on variables, loops and conditionals were completed. During the second to the last week of class, we also gave a post-test to evaluate what the students have learned.

Our goal was to answer these important questions:

1.) Do students who do well on the pre-test also perform well and gain the most from the Codesters Python 1 module?

2.) Do students who use the Codesters lessons have a similar understanding of loops as students in a traditional CS1 course?

3.) Do students who use the Codesters lessons have a similar understanding of conditionals as students in a traditional CS1 course?

Since it was important that we treat all students equally throughout the 15 weeks we spent teaching the coding class, all evaluations -- pre-test, quizzes and post-test - were not graded until after classes ended. At the beginning of the coding class, we had also requested parental permission to use students' data for this research. Only a subgroup of parents gave permission but we did not know who they were until the end of the coding class. Again, we thought that this step was important so that we were not unduly biased towards the actual research subjects.

At our coding class, we did the following Codesters lessons: *Dance Steps, Robot Design, Dialogue, Roll the Dice, Star Variables, My Friends, Math and Computation, Spiral, Variables and Loops, User Input, Intro to If Statements*, and *Tim the Wizard.* Of these lessons, the first seven taught students how to use variables to accomplish a variety of tasks including making a sprite dance across a screen, manipulating shapes which they themselves have named or performing simple mathematical calculations. The next two lessons *Spiral* and *Variables and Loops* taught how loops work. The students created for loops that ran a set number of times. Regrettably, we only managed to get to two lessons on conditions, *Intro to If Statements*, and *Tim the Wizard,* which only covered the idea of if statements. However, we did introduce the students to the idea of an else statement in two unplugged activities.

Of the 25 students in Mr. Greenheck's class, we received permission to include 18 of them in our data set. The 18 students had a good mix of strong, middling and below average students. For the most part, the class was attentive and interested in the material; but as with any class, there were some instances of non-co-operation. For example, some students simply refused to take a quiz. Instead of assigning them a grade of 0, we just removed them from the data set because we cannot determine what they know.

For the topics of loops and conditionals, the students from Central City Cyber School represent our experimental group. They will be compared with the control group which consists of college students from the University of Wisconsin-Milwaukee who took CS250 in Fall 2017 or Spring 2018. CS250 is the first class in a 3-course sequence on programming in Java offered by UWM's Computer Science Department. Among other things, the class teaches variables, loops, and conditionals. The class is often referred to as a CS1 class in the Computer Science education literature.

Most students in CS250 are like the 8th graders from Central City Cyber School – they are learning how to code for the first time. Of the 16 participants in our control group, only two had previous coding experience utilizing a language different from Java. Unlike the 8th graders, however, the CS 250 students are older and further along in their studies. They are intellectually more mature.

Students of CS 250 attend two 50-minute lecture and a 100-minute lab section every week. The course lasts for 14 weeks with four lectures on variables, two lectures on conditionals and four lectures on loops. Since CS 250 teaches programming using Java, not all of the said lectures are relevant to the Codesters lessons which uses Python. In our opinion,

only two lectures on variables, one lecture on conditionals and two lectures on loops are comparable.

During the lab section, a teaching assistant (TA) discusses the lectures from the previous week and answers questions from the students for about 30-40 minutes. The TA gives a quiz for 10-20 minutes and then runs a hands-on lab activity for the last 50 minutes or so. The quizzes are worth 10 points and graded on the basis of understanding of the material, syntax, and cleanliness of code.

The Investigator of this study was a TA for CS 250 during Fall 2017 and Spring 2018. We personally worked with 76 students. An email was sent to each of these 76 students in Fall 2018 asking for permission to use their quiz and exam scores in this study. Sixteen of them agreed. Of this group, 12 are 'A' students, 3 are 'B' students, and 1 is a 'C' student.

Finally, we note that ideally, we want to compare the experimental groups' understanding of variables with that of the control group too but different programming languages handle variables differently. As we already noted, the experimental group was learning how to code through the Codesters Python 1 module. In Python, one can simply create a variable without giving it a type; i.e., one can just declare *x = 3.* The compiler will then infer the type based on the value given and treat it accordingly. However, the control group learned how to code using Java, and in Java one needs to specify the type of the variable. Thus, *x = 3* in Python corresponds to *int x = 3* in Java. This difference between Python and Java affected the way the topic of variables were taught.

The Codesters variables lessons focus on syntax and the general ordering of code. Almost every lesson had students debug code that had an error like attempting to dereference a variable that does not exist or putting double quotes around the variable name.

In the control group, most time was spent understanding the different types variables can have and what can be done between them. For example, students learned the difference between integer division and double integer division or the different methods that can be called on a string variable.

Thus, how Python and Java handled variables affected the way the topic was taught which in turn affected the questions that can be asked. In particular, we could not ask the experimental group a question similar to those asked of the control group because the latter were out of the scope of what was taught in the Codesters Python 1 module.

Section 5: Results

We begin this section by presenting the results of five evaluations that the 18 students from Central City Cyber School took. We then used the data to determine if the pre-test was a good predictor of their performance in the coding class. In Section 5.2, we compare the students' understanding of loops with that of college students who are taking their first programming class (CS1). In Section 5.3, we do the same but this time for conditionals.

Section 5.1: Evaluation Results

We wrote the pre-test with the intent of evaluating the students' skills in logical reasoning, algebraic math skills and coding-related questions. For example, the first two questions required simple math calculations using named variables whose values were provided in the questions. Another asked "If Jeff is at a fork in the road and is 12 years old, [which] path should he take if the sign to go left says [first condition] and the other says [other condition]?" We consulted Mr. Greenheck, the teacher of the class, prior to giving the pre-test, and he concurred that the questions in the test were appropriate for the students and capture the skills we wanted to evaluate. This pre-test was given on the second meeting of the coding club.

The Codesters Python 1 module has seven lessons on variables: *Dance Steps, Robot Design, Dialogue, Roll the Dice, Star Variables, My Friends* and *Math and Computation.* The questions on the variables quiz are similar to the ones the students encountered in their lessons. It was given roughly seven weeks into the coding class.

In a traditional CS1 course, loops are usually taught after the topic of conditionals (i.e. if-else statements). In the Codesters Python I module, the two topics are switched. Interestingly, while the module devotes seven lessons on variables, there are only two on loops which are *Spiral* and *Variables and Loops.* The lessons introduce the concept of "for" loops but not "while" and "do" loops. Nonetheless, the lessons address the many reasons to use loops. Again, the questions on the loop quiz were created based on their lessons. It was given about nine weeks into the coding class.

The Codesters Python I module has four lessons that teach conditionals: *Intro to If Statements*, *Tim the Wizard*, *Begin Rock Paper Scissors,* and *Who Won Rock Paper Scissors*. Unfortunately, due to time constraints our coding class only managed to cover the first two lessons. The questions on the conditionals quiz were based on the lessons and given about 13 weeks into the class.

The post-test was given on the 14th week of class with the goal of testing the students' overall knowledge of coding concepts. The test included questions on variables, loops and conditionals but also on syntax and order of code, etc. Please see the Appendix for a copy of all five evaluations.

For every evaluation, the students were instructed to answer what they could to the best of their ability and that there was no penalty for incorrect or blank answers. The figure below shows the raw scores of all 18 students ordered according to the pre-test scores. We used an "x" when a student did not take the test or was not allowed to take the test. All evaluations were out of ten points.

Pre	Variables	Loops	Condition	Post
9	5	x	2	5
9	9	10	5	4
9	8	10	9	10
9	8	9	7	x
8.5	6	8	7	10
8.5	7	10	9	8
7	4	4	x	x
7	8	8	x	7
7	10	7	8	8
6	8	6	9	2
5.5	3	x	8.5	3
5	10	10	6	3
5	5	6	4	3
4	7	8	4	5
3.5	5	6	x	x
2.5	10	8	8	6
1	2	x	6	x
x	7	7	x	x

Figure 9: Scores on Every Evaluation

The next table contains the descriptive statistics for each evaluation.

Table 1: Descriptive Statistics for Every Evaluation

	Pre-Test	Variables	Loops	Conditionals	Post-Test
Mean	6.615385	6.7777778	7.3125	6.25	5.62308
Median	7	7	8	7	5
Standard Deviation	2.142369	0.563344	2.625992	2.913826	2.719823
Variance	4.589744	5.712418	6.895833	8.490385	7.397436

We note that the summative results of the pre-test, variables, loops and conditionals quizzes were quite consistent and hovered around 70%. The post-test, however, was significantly lower and can likely be attributed to several factors. First, the test had a wider coverage than the individual quizzes and so was harder for the students. No review of concepts took place prior to the test. Second, there was a significant time gap between the start and the end of the coding class. The class had to take one-month break to accommodate the winter breaks of the students and the instructor. Thus, retention of concepts learned may have been an issue. Nonetheless, the results of the three quizzes were quite positive and suggests that overall the students had a good understanding of variables, loops and conditionals after working on the related Codesters lessons.

We also tracked each student's progress throughout the class. In particular, we ask "*Do students who do well on the pre-test also perform well and gain the most from the Codesters Python 1 module?"* To answer this question, we computed each student's rank according to their (i) pre-test score, (ii) quiz average and (iii) quiz and post-test average. A student's rank is equal to the number of people who had a higher score + 1. Thus, the four students whose pre-test scores equal 9 were assigned a rank 1 while the two students whose scores equal 8.5 were assigned a rank of 5. We chose to work with the students' ranks rather than their scores because the latter is influenced by the difficulty of an evaluation. Tracking the students' ranks, however, will allow us to compare a student's performance relative to the others in the class regardless of the evaluation's difficulty. If a student's pre-test rank is lower than their quiz average rank then the student's performance was better than expected. On the other hand, if

their pre-test rank is higher than their quiz average rank then the student's performance was worse than expected.

The rank data is provided below. We excluded the student who did not take the pre-test since there is no basis of comparison for their performance in the class. We also removed students who missed at least two evaluations because we do not have a robust view of their performance in the coding class.

Pre-Rank	Quiz Rank	Quiz Including Post Rank
1	14	13
1	7	8
1	1	1
1	5	5
5	10	5
5	2	2
7	7	6
7	6	3
9	9	9
10	12	11
11	2	7
11	13	12
13	11	10
14	2	4

Figure 10: Quiz Ranks

Below is also a line graph which also displays the above data. Each line is its own color and represents a single student.

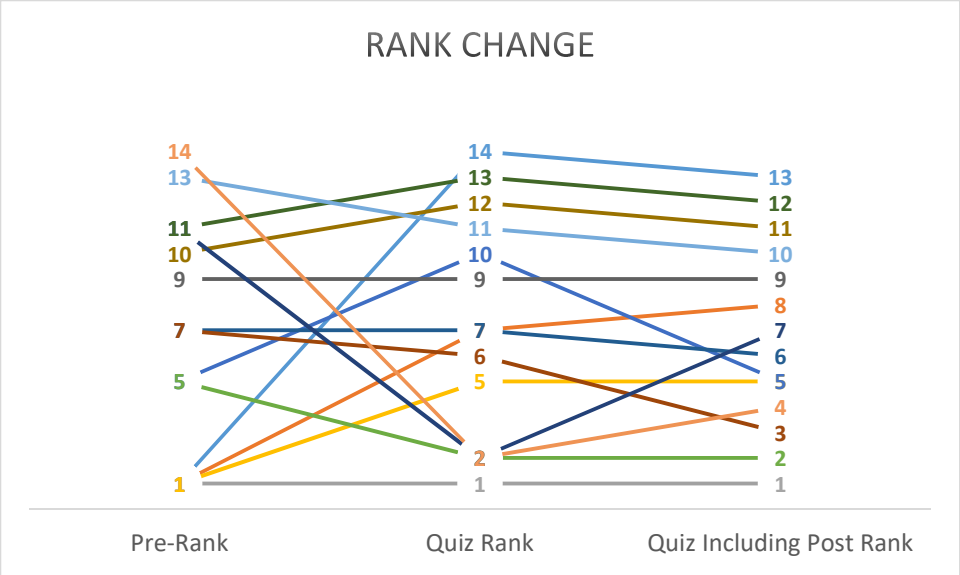

Figure 11: Rank Change

We have a number of students who showed a lot of potential with a high pre-test score but performed poorly in the evaluations. Conversely, there were a couple of students who had significantly low pre-test scores excel in their evaluations indicating tremendous gain from using Codesters. Lastly, we also note that there was a substantial number of students who kept their same work ethic all the way through or improved during the coding class.

To test if the pre-test is a good predictor of the student's performance in the coding class, we ran a linear regression where the pre-test rank (Pre-Rank) was the predictor and quiz average rank was the response. Below is a fitted line plot of the data and the regression coefficients.

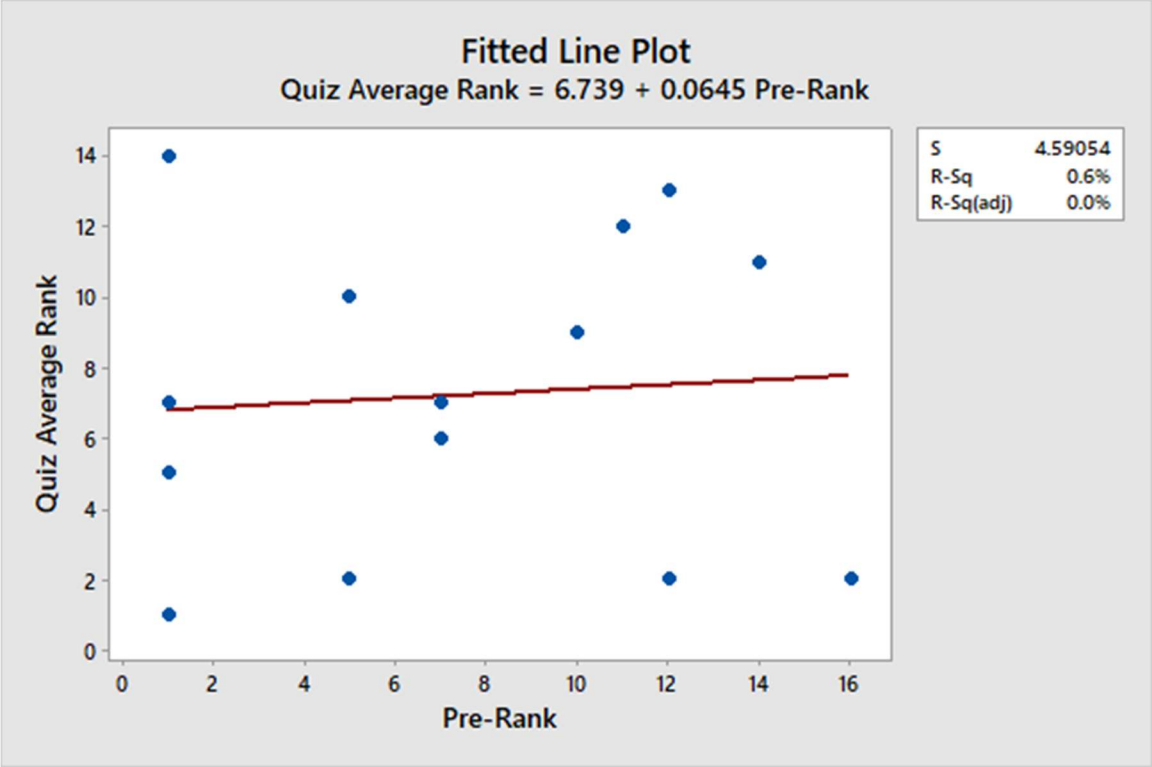

Figure 12: Fitted Line Plot for Pre-Rank v. Quiz Average Rank

Coefficients

Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	6.74	2.17	3.10	0.009	
Pre-Rank	0.065	0.244	0.27	0.795	1.00

Figure 13: Coefficient Display for Linear Regression

To describe a bit more about linear regression, it usually runs based on the following hypothesis test (Weiss, 2008):

$$H_o: \beta_1 = 0 \ (i.e. pre - rank \ is \ not \ a \ useful \ for \ predicting \ quiz \ qverage \ rank)$$

$$H_a: \beta_1 \neq 0 \ (i.e. pre - rank \ is \ useful \ for \ predicting \ quiz \ average \ rank)$$

Since the p-value for pre-rank is 0.795, which is greater than the test p-value of 0.05, we fail to reject the null hypothesis. That is, we conclude that the pre-rank is not a good predictor for the quiz average rank.

To be thorough, below is another linear regression but this time using the quiz and post-test average rank (Quiz/Post Rank) as the response against the predictor of the pre-test rank (Pre-test Rank), along with the same coefficient display:

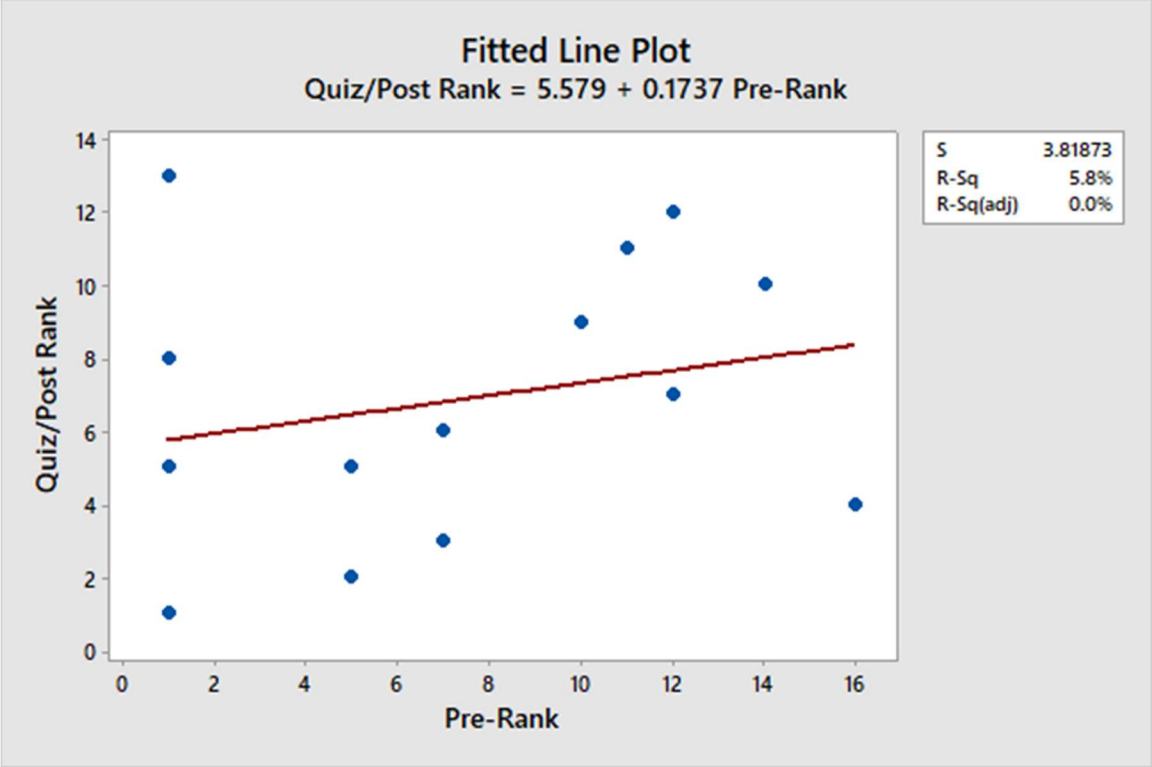

Figure 14: Fitted Line Plot for Pre-Rank v. Quiz/Post Rank

Coefficients

Term	Coef	SE Coef	T-Value	P-Value	VIF
Constant	5.58	1.81	3.09	0.009	
Pre-Rank	0.174	0.203	0.86	0.408	1.00

Figure 15: Coefficient View of Linear Regression 2

Once again, because the p-value is 0.408 (> 0.05), we fail to reject the null hypothesis and conclude that pre-rank is not a useful predictor for the average scores for quizzes and post-test.

These results are quite interesting. They suggest that it is hard to know ahead of time who will learn coding well as evidenced by the two students who ranked first in the pre-test but underperformed in the rest of the evaluations and the two students who ranked 12 and 16 respectively and over performed in the evaluations. This observation brings to mind something that Gordon Smith, the founder of Codesters, said during our interview with him. We were discussing how some students who don't do well in traditional classes seem to blossom in Codesters. "I hear it from literally every teacher in every school. The kids who were excelling at coding were not the kids who excelled in other modalities of learning. So a kid who was sitting in math class, …, doing worksheets and taking tests, was not doing well. But then you [put] them in front of coding terminal and give them the ability to write programs. It's a different skill set…" While we cannot simply attribute the said students' success to Codesters, allowing students to code in a visually rich environment is certainly a very important aspect of the platform.

Section 5.2: Loops

For this subsection, we wish to address the question: "*Do students who use the Codesters lessons have a similar understanding of loops as students in a traditional CS1 course?*"

As we already noted, students in the experimental group were given a quiz after the loops lessons were completed. Of the five questions in the quiz, we considered the results of only two questions because similar questions were asked of the control group. The questions for the experimental group are as follows:

```
1E) How many times will Phil say "Good Morning Class!" If the command was in the following loop:

for counter in range(5):
    Phil.say("Good Morning Class!")

2E) What will be the final result of x?

x = 3
for counter in range(10):
    #Add 3 to x each time
```

Figure 16: Questions asked of Experimental Group - Loops

Each question was worth 5 points. The control group were asked questions like these:

```
1C) Write a Java while loop which prints to the screen: ALL integers from 100 down to 80

2C) Produce a "Short form trace/run" of the program below:
int s = 0;
int i = 3;
while(i < 6)
{
    s = s + i;
    i = i + 1;
}
System.out.println("s = " + s);
```

Figure 17: Questions asked of Control Group - Loops

Students from the same lab section were asked the same questions but students from different lab sections were asked similar but different questions to prevent cheating. For example, the numbers 100 and 80 in question 1C may be replaced by the numbers 200 and 160, etc. Again, each question was worth five points.

We note that 1C and 2C made use of “while” loops but they could easily have been modified to use “for” loops as shown below:

```
1C*) Write a Java for loop which prints to the screen: ALL integers from 100 down to 80

2C*) Produce a "Short form trace/run" of the program below:
int s = 0;
for(int i = 3; i < 6; i = i + 1)
{
    s = s + i;
}
System.out.println("s = " + s);
```

Figure 18: Questions for Control but with for loops

All four questions that we considered capture the idea that a loop repeats an action. Question 1E simply asks a student how many times the loop will repeat. For question 2E, the student must also take into account the addition step that modifies the value of *x* within the loop. Question 1C gives the student a hint on the number of times the loop should repeat, but the student must also write the appropriate code. Question 2C is very similar to question 2E except that the number added to *s* changes from one iteration to the next. We acknowledge that questions 1C and 2C are harder than questions 1E and 2E respectively. But they account for the fact that the students in the control group are intellectually more mature than the students in the experimental group. The underlying ideas between 1E and 1C and between 2E and 2C, however, are quite similar. Consequently, we argue that it is fair to compare the results of the two groups.

Out of the 18 participants in the experimental group, two participants did not take the quiz and were unable to make it up and one was caught cheating during the quiz. Thus, we only

have 15 data points for this group. All 16 participants' scores in the control group were available for analysis. The figure below shows the score distribution of the two groups.

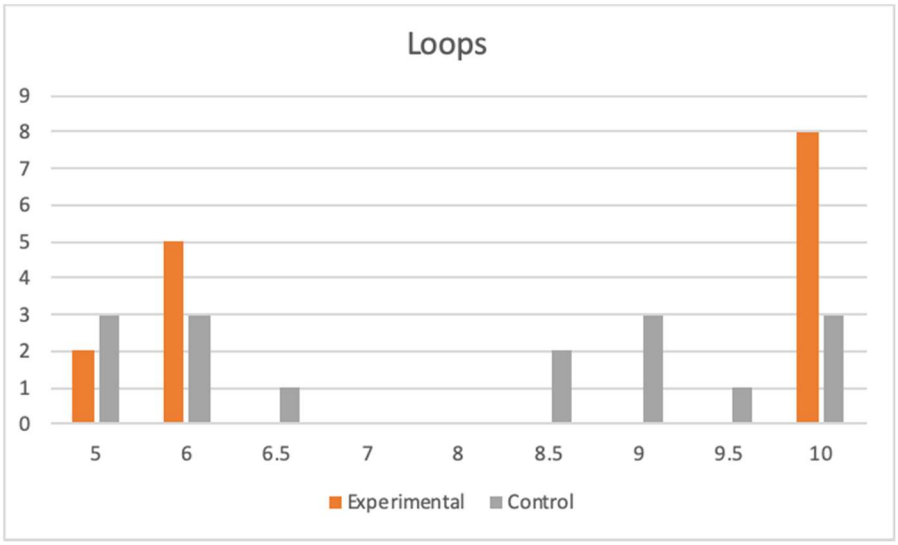

Figure 19: Grade Distribution for Loops

We can see that the score distribution for both groups are quite similar. Most students either got full points or half of the points. To better understand our data, we ran descriptive statistics in Excel and found the values listed below:

Table 2: Descriptive Statistics for Loops

	Experimental	Control
# of data points	15	16
Mean	8	7.6875

Median	10	8.5
Standard Deviation	0.57735	0.489206
Variance	5	3.829167

Below is our null and alternative hypotheses for statistical testing; we expect to reject the null hypothesis.

$$H_o: \mu_E = \mu_C$$

$$H_a: \mu_E \neq \mu_C$$

To compare the mean scores of the experimental and control groups, we first consider the pooled t-test. Two assumptions must be met: the experimental and control data sets are independent and they both have a normal distribution. If a data set has 30 or more points, it is generally safe to assume normality [GZ12]. But our experimental and control groups have 15 and 16 participants respectively so we must check if the scores distributions are in fact normal. We used the Shapiro-Wilk test on our experimental group and obtained the following results using Excel:

n	15				
				diff	a*diff
a1	0.515		x15-x1	5	2.575
a2	0.3306		x14-x2	5	1.653
a3	0.2495		x13-x3	4	0.998
a4	0.1878		x12-x4	4	0.7512
a5	0.1353		x11-x5	4	0.5412
a6	0.088		x10-x6	4	0.352
a7	0.0433		x9-x7	4	0.1732
					7.0436
	SS	70			
	b	7.0436			
	W=b^2/SS	0.708747			
	0	0			
	0.01	0.835			
	p-value	0.008485			

Figure 20: Shapiro Wilk performed on Experimental

Our test value for the Shapiro-Wilk test is 0.05. The p-value of the experimental group's score distribution is 0.008485, which is smaller than 0.05. Thus, we conclude our experimental group's data is not normal. To double check our calculations, we ran the Ryan-Joiner test on Minitab, a normality test very similar to Shapiro-Wilk, and found the following:

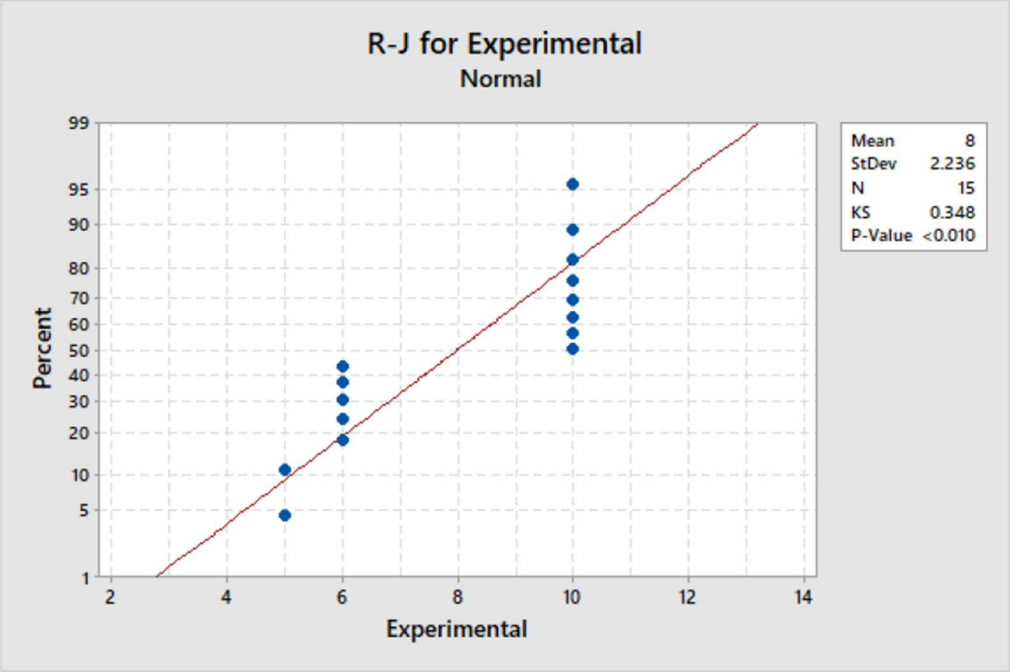

Figure 21: Ryan Joiner done on Experimental Group

Again, our test value for the Ryan-Joiner test is 0.05. The p-value of the experimental group's score distribution under this test is less than 0.010, which is again less than 0.05. The Ryan-Joiner test again confirms that the experimental group's dataset is not normal.

However, despite the fact that the t-test assumes normality, it has been shown to remain robust when the experimental and control groups' data sets are roughly equal in size [P84]. Since our data sets differ by only one participant, we decided to simply run the pooled t-test. Below is the result obtained through Minitab:

Test

Null hypothesis	$H_0: \mu_1 - \mu_2 = 0$
Alternative hypothesis	$H_1: \mu_1 - \mu_2 \neq 0$

T-Value	DF	P-Value
0.41	27	0.683

Figure 22: Pooled T-test with Non-normal data

Since the p-value of 0.683 is greater than our confidence interval of 0.05, we fail to reject the null hypothesis.

But our experimental group's data does not have a normal distribution so we must utilize one of the non-parametric methods for testing. One of the most popular methods is the Mann-Whitney U-Test. Unlike the t-test and tests like it, non-parametric tests consider the dataset's medians. The Mann-Whitney test has been shown to be just a little less robust than the t-test yet a good alternative to it [W08]. Below is the result of running the Mann-Whitney test on our dataset.

Mann-Whitney Test and CI: Experimental, Control

```
               N  Median
Experimental  15  10.000
Control       16   8.500


Point estimate for η1 - η2 is 0.000
95.4 Percent CI for η1 - η2 is (-0.999,1.500)
W = 261.5
Test of η1 = η2 vs η1 ≠ η2 is significant at 0.4065
The test is significant at 0.3903 (adjusted for ties)
```

Figure 23: Mann-Whitney test results for Loops

The resulting p-value is 0.3903, which is greater than our test p-value of 0.05. Thus, we fail to reject the null hypothesis. This test concludes that there is no significant difference

between the experimental and control groups' scores. That is, the students who learned loops via the two Codesters' lessons understood the concepts of loops as well as the college students who learned loops in a traditional CS1 class.

Section 5.3: Conditionals

For this subsection, we wish to address this question: *Do students who use the Codesters lessons have a similar understanding of conditionals as students in a traditional CS1 course?*

Of the five questions asked in the conditionals quiz, we included the result of only one question in our study because comparable questions were asked of the participants in the control group. Below is the question used for experimental group; it was worth 5 points.

```
1E) What will happen if the value of name is John?

if name == "James":
    Sprite.say("hello!")
if name == "John":
    Sprite.say("goodbye!")
```

Figure 24: Experimental Group Question - Conditionals

The control group was asked a question similar to the one below. Again it was worth 5 points.

```
1C) Produce a "Short form trace/run" of the program segment below:

int i = 0;
if(i>=0)
{
    System.out.println("Non-negative");
}
else
{
    System.out.println("Negative");
}
```

Figure 25: Control group question - Conditionals

Both questions 1E and 1C tests a student's knowledge of branching code and their ability to trace code to determine its output. Although question 1E uses an if statement only, it is easy to modify the question so that it utilizes an else or an elif statement like question 1C.

```
1E*) What will happen if the value of name is John?

if name == "James":
    Sprite.say("hello!")
else:
    Sprite.say("goodbye!")


1E**) What will happen if the value of name is John?

if name == "James":
    Sprite.say("hello!")
elif name == "John":
    Sprite.say("goodbye!")
```

Figure 26: Experimental group question written with Else

Of the 18 participants in our experimental group, we only included that scores of 13 participants because four of them refused to take the quiz and one left it all blank. Of the 16 participants in the control group, only 9 of them took a quiz that had question 1C in it. The scores distributions are shown below followed by their descriptive statistics.

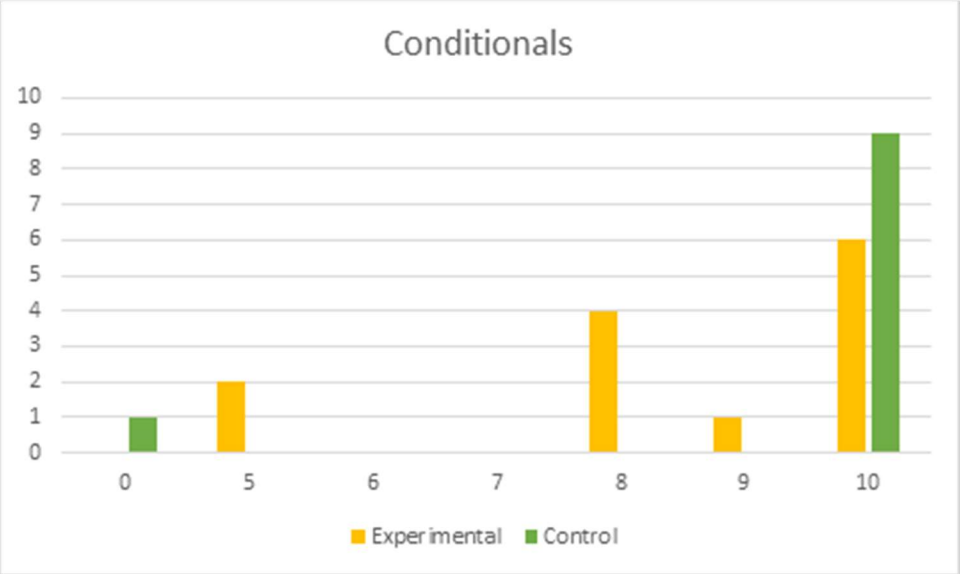

Figure 27: Grade Distribution for Conditionals

Table 3: Descriptive Statistics for Conditionals

	Experimental	Control
Mean	8.538462	8.75
Median	9	10
Standard Deviation	1.808101	3.535534
Variance	3.269231	12.5

Despite the fact that we did not complete the four lessons on conditionals in the Codesters Python 1 module, we suspect that the experimental group's understanding of conditionals is on par with that of the control group. Below is our hypothesis:

$$H_o: \mu_E = \mu_C$$

$$H_a: \mu_E \neq \mu_C$$

Unlike Section 5.2, we want to utilize the separate t-test as the ratio of variances does not fall in the range of 0.5 to 2 but have to first begin by checking the normality of the experimental group's scores distribution. Below is the result of running the Shapiro-Wilk test in Excel on the experimental group:

n	13				
				diff	a*diff
a1	0.5359		x13-x1	5	2.6795
a2	0.3325		x12-x2	5	1.6625
a3	0.2412		x11-x3	2	0.4824
a4	0.1707		x10-x4	2	0.3414
a5	0.1099		x9-x5	2	0.2198
a6	0.0539		x8-x6	2	0.1078
					5.4934
	SS	39.23077			
	b	5.4934			
	W=b^2/SS	0.769229			
	0	0			
	0.01	0.814			
	p-value	0.00945			

Figure 28: Shapiro Wilk performed on Experimental - Conditionals

The p-value of the dataset is 0.00945, which is less than our test p-value of 0.05, so we conclude that the data set is not normally distributed. Despite the fact that we do not have normality we still ran the pooled t-test to see what the result will be. Below is what we obtained through Minitab:

Test

Null hypothesis	$H_0: \mu_1 - \mu_2 = 0$
Alternative hypothesis	$H_1: \mu_1 - \mu_2 \neq 0$

T-Value	DF	P-Value
0.48	12	0.639

Figure 29: Separate Variance T-Test without Normality - Conditionals

The p-value of 0.639 implies there is no significant difference between the means of the scores distribution of the experimental and control groups. But we do not have normal data so we ran the Mann-Whitney test on the two datasets. Below is the result.

Test

Null hypothesis	$H_0: \eta_1 - \eta_2 = 0$
Alternative hypothesis	$H_1: \eta_1 - \eta_2 \neq 0$

Method	W-Value	P-Value
Not adjusted for ties	156.00	0.689
Adjusted for ties	156.00	0.674

Figure 30: Mann-Whitney results - Conditionals

The p-value of 0.674 is in line with the findings of the t-test -- there is no significant difference between the score distributions of the experimental and control groups. However, due to the simplicity of the question given to the control group, we wanted to consider a harder conditionals question for comparison. We refer to this question as 2C.

```
2C) Given the following complete Java program, produce a "Short form trace/run"
assuming the input value is 6

Scanner stdIn = new Scanner(System.in);
int x = 3;
int y = 18;

System.out.println("x = "+x+", y = "+y);

System.out.println("Please enter a whole number:");
x = stdIn.nextInt(); //6
System.out.println("x = "+x+", y = "+y);
if(x>y)
{
    System.out.println("x>y");
}
else
{
    System.out.println("x<=y");
    if(x%2==0)
    {
        System.out.println("x is even");
        if(y%2==0)
        {
            System.out.println("y is even");
        }
    }
    else
    {
        System.out.println("x is odd");
    }
}
```

Figure 31: Control Group Question 2C

Question 2C is harder than question 1C. The code contains multiple nested conditionals so figuring out the output is more challenging. But we feel that comparing the results of 1E to 2C is fairer given that the experimental group consists of middle school students while the control group consists of college students. Below are the scores distribution for questions 1E and 2C:

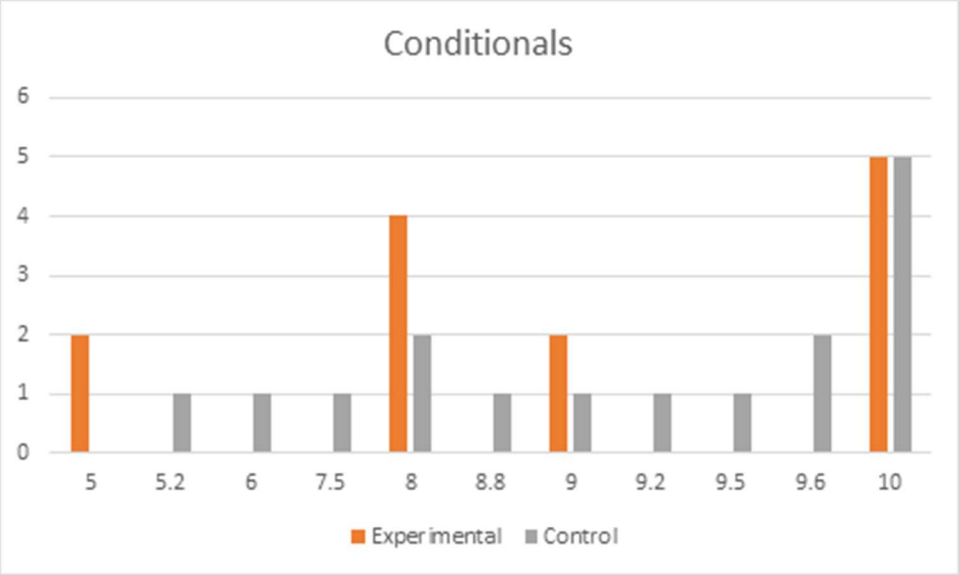

Figure 32: Grade Distribution Utilizing 2C

Again to better view our data, see the descriptive statistics in the table below:

Table 4: Descriptive Statistics for Conditionals using 2C

	Experimental	Control
Mean	8.538462	8.775
Median	9	9.35
Standard Deviation	1.808101	1.481666
Variance	3.269231	2.195333

As we can see the scores distribution for question 2C is much more spread out. Since we've already confirmed that the experimental group is not normally distributed, we simply

present the test results. Below is the result of the pooled t-test for the scores distribution of questions 1E and 2C:

Test

Null hypothesis	$H_0: \mu_1 - \mu_2 = 0$
Alternative hypothesis	$H_1: \mu_1 - \mu_2 \neq 0$

T-Value	DF	P-Value
-0.38	23	0.708

Figure 33: T-test on Conditionals using 2C

Once again, the p-value of 0.708 is greater than the test p-value of 0.5. But because the data is not normal, we also ran the Mann-Whitney test. Below is the result:

Test

Null hypothesis	$H_0: \eta_1 - \eta_2 = 0$
Alternative hypothesis	$H_1: \eta_1 - \eta_2 \neq 0$

Method	W-Value	P-Value
Not adjusted for ties	194.50	1.000
Adjusted for ties	194.50	1.000

Figure 34: Mann-Whitney results for Conditionals using 2C

The p-value of 1.0 matches the finding from the t-test -- there is no significant difference between the two medians of the data. At this point, we feel it is safe to say that students who learned conditionals using two lessons from Python 1 understood the concept of the if-conditional as well as the students who learned conditionals through a traditional CS1 class. The conclusion confirms our initial hypothesis.

Section 6: Conclusion and Future Work

We had set out to answer the question of whether or not the Codesters Python 1 module was effective in teaching the basic coding concepts of variables, loops and conditionals. We ran a study where we taught coding to a class of eighth graders at Central City Cyber School using Codesters for 15 weeks. The students took a pre-test to establish a baseline for their problem-solving and math abilities. They then had quizzes after the Codesters lessons for variables, loops and conditionals were completed. Towards the end of the coding class, they also took a post-test that covered all three concepts. The students' average and median scores for the quizzes were around 70% which suggest they had a good understanding of the concepts. Their post-test scores though were lower. We then ran linear regressions to determine if the pre-test was a good predictor of the students' performance. Surprisingly, the answer is "no." Some students who had high pre-test scores underperformed while some students who had poor pre-test scores did really well during the coding class. The founder of Codesters, Gordon Smith, has expressed similar observations. He attributed it to the fact that Codesters offers a different modality of learning that seems to resonate with some students who don't perform well in a normal middle school class.

We also wanted to know whether students who learned coding through the Codesters Python 1 module understood the concepts of loops and conditionals as well as students who learned coding the traditional way. Our experimental group consisted of the students from Central City Cyber School. Our control group was made up of UW-Milwaukee students who took CS 250, the first programming class in a sequence of three taught by the Computer Science Department. We considered results of specific questions from the quizzes the experimental

group took during coding class and compared them against results of similar and sometimes harder questions from the quizzes and exams that the control group took during the CS 250 class. For both concepts, we found that our experimental group fared as well as our control group. That is, for this specific group of students, the Codesters Python 1 module is as effective as a traditional CS1 class (i.e., a beginner programming class) in teaching loops and conditionals.

There are a few points we would like to make about our study. First, we had planned to do a comparison of the experimental and control group's performance for variables too but we regretfully were not able to find common ground between our experimental and control groups. The experimental group learned coding using Python while the control group used Java. The two programming languages treated variables differently which affected the way the concept was taught. In the future, we hope to work with a new control group that uses Python so we can do a valid comparison.

Second, we note that the questions we used in Sections 5.2 and 5.3 gives the impression that we were just testing the experimental group's ability to read and analyze code. But, in fact, we also asked them to write code. In the conditionals quiz, one question asked students to make an if statement which checked if a variable named number was equal to four. The experimental group fared very well on this question with only a few minor syntax errors. The other question is shown below. In retrospect, it is harder and more ambiguous than what we intended.

```
5) Let's imagine a game of rock, paper, scissors. Write the if statement that would
check the situation where a user chooses rock and make another right under it
where you check if the user chooses paper.
(Name the variables whatever you'd like and assume rock and paper are Strings)
```

Figure 35: Last Conditional Question

We expected the answer to this question to look something like this:

```
if choice == "rock":
    #do something
if choice == "paper":
    #do something
```

Figure 36: Expected Answer

Not surprisingly, the students struggled to formulate an answer. This is a question that a typical CS1 student can answer but was a bit beyond the capability of experimental group. Upon consulting the revised Bloom's taxonomy, this was more than likely due to us asking them a question of the Create knowledge level when they had only had experience with skills of a lower knowledge level [AEA01].

We had also planned to compare the results of these two questions against something done by the control group. However, the control group was never asked to simply write an if statement. They wrote if statements as part of an answer to a larger question. Thus, the scores they received were not a true reflection of their ability to write if statements.

As for loops, there were not many instances in which the students in the experimental group had to write a loop. Usually, whenever they need a loop in their Codesters lesson, they would drag out a code snippet from the logic tab and modify it.

We did encourage them to type out the loops but only a few students attempted to do so. Hence, there was no fair way to test student's ability to create loops against the control group.

Third, one criticism we will likely receive from our research study is that we are comparing the performance of two very different groups. Since we did not have the opportunity to run a traditional CS1 class for middle school students, we had to use college students. It goes without saying that latter are more intellectually capable than the middle school students. We tried to alleviate this advantage by considering harder questions for the college students to arrive at a fair comparison.

Fourth, another likely criticism is that our sample size was small and doesn't represent the population as a whole. Indeed, a bigger study is needed. We are continuing our research at another school in the Milwaukee area. We are just in the beginning stage at the time of writing and plan to incorporate the results we obtain to our current study.

Lastly, our study found that the Codesters Python 1 was effective in teaching our experimental group the basics of coding. Despite the lack of interest or exhaustion displayed by some towards the end of our program, we truly feel that with the right material, the right teacher, and the right lesson plan that Codesters has the ability to reach many different types of students. As we continue working with schools in the local area, we will always be looking at ways to improve teaching programming. As for Codesters, Mr. Smith has hinted that new modules and ideas are on the way, so we look forward to the innovation this relatively young company will bring to the table.

References

[A95] Alice. 1995. *alice.org*

[AMB15] Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" Programming. *ACM Transactions on Computing Education*.

[ACM/IEEE13]Association for Computing Machinery (ACM) and IEEE Computer Society. (2013). Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. 169.

[AEA01] Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E. , Pintrich, P. R. , … & Winrock, M. C. (2001). A Taxonomy for Learning, Teaching, and Assessing: A revision of Bloom's Taxonomy of Educational Objectives. *White Plains, NY: Longman.*

[C14] Codesters. 2014. *Codesters.com*

[CB14] The College Board: AP Computer Science A. 2014
https://apcentral.collegeboard.org/courses/ap-computer-science-a/course

[GZ12] Ghasemi, A., & Zahediasl, S. (2012). Normality Tests for Statistical Analysis: A Guide for Non-Statisticians. *International Journal of Endocrinology and Metabolism*, 486-489.

[GB17] Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. *SIGCSE*, (pp. 267-272). Seattle.

[HA17] Hermans, F., & Aivaloglou, E. (2017). Teaching Software Engineering Principles to K-12 Students: A MOOC on Scratch. *International Conference on Software Engineering: Software Engineering and Education Track* (pp. 13-22). Buenos Aires: IEEE Press.

[MBZ18] Mladenovic, M., Boljat, I., & Zanko, Z. (2018). Comparing Loops Misconceptions in Block-Based and Text-Based Programming Languages at the K-12 Level. *Education and Information Technologies*, 1483-1500.

[PD17] Parker, M. and DeLyser, L. (2017). Concepts and Practices: Designing and Developing Modern K-12 CS Framework. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 453-458.

[P84] Posten, H. (1984). Robustness of the Two-Sample T-Test. *Robustness of Statistical Methods and Nonparametric Statistics,* 92-99

[S07] Scratch. 2007. *scratch.com*

[TC18] Topalli, D. and Cagiltay, E. N. (2018). Improving Programming Skills in Engineering Education Through Problem-Based Game Projects with Scratch. *Computers and Education*, 64-74.

[W08] Weiss, N. (2008). *Introductory Statistics.* Boston: Pearson.

[DPI] Wisconsin Department of Public Instruction. https://dpi.wi.gov/computer-science

Appendices

A: Interview with Gordon Smith, CEO and Founder of Codesters

Antonio: So I was just hoping to ask you a few questions about your background and about Codesters in general.

Gordon: Can you give me a reminder really quickly about what your program is and what you're working on, so I can have a reminder of what the context for this is?

A: Sure! So currently I am doing my Master's thesis on the effectiveness of Codesters for translating over to actual text programming for specific topics such as variables, loops, and conditionals. I'm testing to see when it's done in the Codesters environment if the students can also display that on paper.

G: Wow! That's awesome. What stage are you in the research?

A: Right now I am in the data collection [stage]. I'm working at a local Milwaukee middle school and giving them assessments as we go along and sticking to Codesters Intro to Python I tutorial.

G: That sounds great I would love to see what the results are in terms of the data and in terms of recommendations. I mean that's fantastic research we'd love to see more of done. So I appreciate that.

A: Yeah not a problem! I will definitely let you know as more results come in and especially once it is published. Especially since I probably need your permission to use Codesters name and everything.

G: Makes sense, yeah absolutely.

A: Perfect! So um, I'm just gonna begin with a few background questions, what is your background in Computer Science and Programming and what inspired you to create Codesters?

G: Yeah so my background combines really three things. One is a background in education, I've run education companies my whole professional career for the last 18 years. Starting with a tutoring company I founded in 2001. I also have my sort of educational background because I have undergraduate degree in engineering and Masters of Business Administration. So really education, entrepreneurship, and engineering came together at this critical moment to form Codesters. So my background with coding is really from the engineering background from my undergraduate days which was 20 odd years ago but it served as a model for me to think about how kids learn but also how they can use coding to kind of explore the thinking of how we approach using coding as kind of a tool to be used to then explore topics in math and science and used as sort of a modeling tool and not really focused on things like robotics or web development that might be more [things you would do if you] were a professional programmer. For me it was more going back to its roots of being used as a tool for modeling engineering simulations and things like that. So it informed a little bit of that background but about five years ago when we were involved with my other companies we really saw demand coming up rather suddenly with students, with families, with schools and we really saw that people wanted to code but nobody seemed able to do it and private schools were having a hard time getting students learning coding. Kids felt like they were going to college unprepared for CS even though we were sort of promoting this. Well I've done a lot of education in that middle school/ high school age, I know what I think would be a good structure for a curriculum for

working with this kids for producing their artifacts from their projects and sort of have an idea of how we can make something that really scale and can really reach 50 million kids. So the vision was, what do we need to do to create a coding program that can reach 50 million students in the K-12 population [that] is ideal for what we are trying to create. So the decisions we made very early on I think have been very prescient and maybe even controversial when we made them but I think was the right thing to do. One of them was we chose to make a tool for classroom teachers and I can tell you in 2014 when we started Codesters, the idea that we were going to go into a middle school and have existing experienced middle school teachers teaching programming, I can tell you that most people in the CS space were like "yeah that's not gonna happen" but I have pretty good faith in educators and teachers to do this and I think what we are asking them to do isn't beyond them. I think we give them the right tools, the right curriculum, the right support so they can actually do this. Partnering with teachers, partnering with schools, putting this during the school day rather than an after school program so it [is] really part of the core academics, those were some of the early decisions about the way that we approached this problem of "how do we get kids coding" that went into the underlying design that went into making it. And then, ideas like we wanted them to learn a real language because we thought by middle school they should be able to do that. We picked Python which is a really good choice, it's a really simple language and is a really popular one. And one that has applications other than you can make a web app. Python is more than likely the language you are using if you are a scientist and want to do data analysis. They're probably going to have you use Python if you are doing major data analysis on the work you're doing, it has this really broad realm of possibilities. It's not just well we're choosing Swift so they can make a mobile

app. We are teaching them a foundational language that is applicable in a variety of career domains whether or not they want to do something strictly technical or just something that uses something technical. So long answer, but that's the foundational roots and ideas that went into making Codesters.

A: Excellent! And yeah, I definitely agree with you that Python was the best way to go. It's a language that somebody with no background can just sort of jump in and figure out. So with that in mind though, I actually have a question for you about your curriculum. So in your Python I curriculum, you introduce loops before conditionals. Now in a traditional CS course, it would usually be opposite. So what made you present those first?

G: [laughs]. Well when we first started out we threw out the traditional. I mean most things would start with you know data types but we thought, hey we're building this for 10, 11, 12 year olds what do we want them to do? So the first thing we want them to do is make a visual project and we start to think about the rational steps taken for them to actually build something. The reason we put conditionals where we did was because we wanted there to be a reason why there would be branching behavior. So in most curriculum when you don't have a method of taking in user input or you don't have another way to have a meaningful option in your program then you're basically hard coding the condition into your program yourself. So you're saying if my answer is greater than 10 do this, if not do that. Well if I have to put answer equals 10 into my program, to a 12 year old that's like well why hard code the solution before the if statement, so we needed to have this happen at a certain point where they've already done something with variables so they can store a response and something where they ask a user for a response before we could come up with a meaningful context for them to use an if

statement in a way that would matter to an 11 year old. Until you can ask a question, collect an answer, and store it can you use it in an if statement in a meaningful way. So we looked at it as how do we create a sort of sequence of skills in a logical progression that [a student can relate to], rather than what is the traditional thinking of the order of the topics.

A: Ok, alright, makes sense.

G: So we are thinking about it pretty basic, we almost approached this a little bit from a developmental standpoint thinking about students. And I think [about] teaching looping behavior to a five year old but the necessary machinery [in] an if statements not too bad but if we're doing it in the context of reading variables and [they] are a more complicated concept we have to do more. So I would say if we get to a point where there is a K-12 scoping sequence of skill progression I think you will find loops coming in before if statements and both coming before variables. I think we'll see an emergence of at what point are students developmentally able to handle certain topics of abstraction concepts.

A: Alright. So my next question is, obviously in this field of computer science education there are a lot of companies vying to get into this [field] and [many] know there are the big names like Scratch and Alice. So in your opinion, what separates you from those other names? What do you think Codesters has that the competitors don't?

G: I think our take on how we can use coding as a foundational skill, or even a literacy for students, as part of their core academics is more advanced than a lot of people are thinking. Where they're thinking that this is about some separate thing of either computational thinking or it's about training them to be a developer or coders and I'm really looking at this skill as a

way of giving these kids a toolkit [that] they can [then] go into a math class and create a cool project in statistics [for example] that they couldn't have made without it or they can make a simulation in a science class and create a simulation for something they'd otherwise have to describe [out loud] or write an essay about. We're giving them this literacy that is then part of the way they look at and explore the rest of STEM and we really think of this as the glue that binds them together and when I look at it I see a lot of people in the CS education space are very much in the silo of computer science and we've really tried to push into the core of academics as making a class that every kid has to take, not just some elective where some of the kids do it or some after school club. We want this to be taught by regular classroom instructors not by volunteers coming in from outside. This should be tied into academic achievement in all core subjects not in its own little separate side thing. That approach and the way we've gone about that, I think [is] informed a little bit by my background in education. I think that makes it unique from other people who are very much siloed into CS as a separate standalone space because I think they're missing very important ways that it has to be part of the larger school community and larger school academic endeavor in order for it to really take hold.

A: For sure and there's been countless studies that have shown that if students are exposed to this kind of projects [and] this kind of thinking. The results show that in other science fields that the thinking that is used in computer science is going to help you in no matter what ever field you decide to go into science wise.

G: In general, that domain transfer you are describing has been fleeting and difficult to prove in most of the studies I've looked at. Basically, the idea that they can do coding will develop

problem solving skills that'll magically transfer to math and science has been a vision for years but has not really been shown true. I think it only works if you directly connect them. If you take coding into the math class or into the science class. I think that domain transfer problem has been proven to be much much harder. I think most studies have shown that just doing coding in and of itself drives, in the causal way, doesn't necessarily drive achievement in math. Particularly in math, I always see it in math, I haven't seen it in much about science. But there are some studies that are trying to show that if kids do coding they'll get better at math and it's not a direct domain transfer. You really have to do them in conjunction as a combined skill for that transfer to happen. I think a good place to look for this as research is Bootstrap by Emanuel Schnazer, you can read his PhD dissertation. His curricula is one of the ones that has demonstrated because he does coding in algebra that puts thing in contexts of algebra [using] things like functions and domain and range but he has a good lit[erature] review on all the studies that tried to tie coding to math achievement and the difficulty or failure to show that domain transfer. You'll often hear people praise Emanuel of having basically the only curriculum that can really show the increase in math scores. So that's one of the things that people are going to have to understand and recognize that at a certain point it won't be magic. Coding will not magically give them [math] skills, it won't magically help, it HAS to be a direct integration of coding in math and science for those skills to show up and effect what students are going to do. And I think that's what we're going to find over the next couple of years, [once the] studies and the standard alignment and the outcomes really start to come in. I think right now what they're looking at unfortunately is some pretty broad surveys that are basically

showing that the kids who were good at coding were the ones who were good at math but that's what [they're] are finding now as opposed to them affecting actual performance.

A: Yeah I think it's one of those case by case things. It's actually really interesting, this is actually my second semester teaching this at this specific middle school and my previous semester I did have a student who was kind of falling behind in things but eventually just through the help of Codesters and really buckling down and really doing all the lessons here, he showed tremendous progress. Of course that could've been just for him. He could've been just having fun and saying hey how can I use this for anything else?

G: Well this would be interesting to study because I can tell you it is so pervasive in our anecdotal evidence but I do not have any data to prove it and [it is] difficult to prove. But I hear it from literally every teacher in every school. The kids who were excelling at coding were not the kids who excelled in other modalities of learning. So a kid who was sitting in math class, listening to the teacher, doing worksheets and taking tests, was not doing well but then you [put] them in front of a coding terminal and give them the ability to write programs it's a different skill set. It's a different way of learning and actually it pulls a different group of kids into the acceleration. The most glaring examples are in the special need classrooms, I've had people tell me they've had non-verbal students who have become classroom readers because of their coding ability, who had previously basically been excluded from the class. So I think it'd be a really interesting study to look at because there is a fundamentally different modality to the learning in a coding platform. We talk about active learning and project based learning but you can see it. You walk into a coding classroom again where they are doing math in a coding platform as opposed to doing math it is a fundamentally different image, sound, and behavior

pattern in the classroom and it turns out it appeals to a different group of students. One of the things looking at is how to help reach and attract and retain students that are otherwise struggling and then I think it becomes [one] of these things where coding pulls in these kids that didn't like sitting in a lecture. Maybe they don't take in auditory information, maybe they don't raise their hand for the teacher. Maybe they're embarrassed or shy. Maybe they don't do well in a timed situation. But you put them in front of something that requires them to be meticulous problem solvers where because they can run a program [and] run it right away and see if they did it right or wrong and they can debug it themselves and then turn it in when they're done at their own pace and it turns out that they are fantastic at that. Its perhaps a better skill to have in real life because it looks better [to] a regular job than taking a quick timed math test and it turns out you are really appealing to a different modality of learning [that] is demonstrated knowledge that is not captured in a current classroom environment. I think that is one of the most powerful things of the coding phenomenon because it may just [get] kids interested in learning because it's giving them this whole different way of acquiring information and being evaluated that is really appealing to them.

A: Alright. Just to kind of get through some questions here.

G: Sorry I give long answers! I talk about this a lot! [laughs]

A: No worries! And hey that's alright, what I need is some good content [and you are providing that]! So now that you've seen your software in action. Now that you've deployed it and I saw that you guys are in a lot of New York Tier 1 schools, is there anything that you'd like to change or add to your existing software?

G: I can give you a little preview of what's coming, we're sort of beta testing something right now. What you are working with right now is our sequential course offerings [of] our Python I and II but I think we initially imagined that there was going to be a 3 and a 4 but I think what we are doing instead is reconfiguring that to be a grade level curriculum so we have a 6th-8th grade separate curriculum and we've integrated more of the math and science project work directly into the pathway. So we have a cadence where they'll do 3 coding lessons and then they'd have a project where if they were in a science class, the teacher could embed a lab where they use these skills and tie more stuff into it and use this coding to explore and demonstrate these math and science concepts in a sort of lab context. Directly within that curricula and sort of being supported by grade level aligned rather than sequential. So if you are doing that in 6th grade, we have 6th grade labs directly embedded into it. And the next piece of it is working down to other younger grades where people are trying to use our platform are beginning to recognize that kids like 4th graders really struggle with things like the coordinate plane and it's not really about coding but giving them a 4 quadrant coordinate plane with x's and y's up to 250,-250 they've never seen that before. So giving them an environment that is more developmentally appropriate, more age appropriate, making it a little more blockish so really going through a fine gradation of what the environment should be at each grade level and having the distinct grade level curriculum along with integrating the math and science topics into the curriculum is one of the things that we are working on right now.

A: Excellent! So you might've mentioned this already but what do you think is the significance of – cause you know in Scratch it's essentially pseudocode like while this do this – do you think

kids are gaining more from seeing the actual Python code? Opposed to using almost a pseudocode?

G: I think it's a little bit about – well, I actually think blocks are frustrating kids. I think the things we see with kids at this age is if the gain proficiency the idea of going back to a toolkit and dragging them out as opposed to being able to freely create their code. It's like telling kids to write an essay with refrigerator magnets [and] at a certain point it's like can I just write? I think kids might actually prefer it a little bit. And also then things like variables aren't in dropdown they're not these blocks you just drag around, they're just writing in text. This is an age where we expect them to write essays and we expect them to communicate in this written format so this just matches what they are doing developmentally [and] it would be very frustrating to try and write a program in this sort of block environment so I think they do in fact prefer this modality. I think when they get used to having this set of blocks they just stick with them and they just pop it together and if it works so be it. And I think this idea where you actually get errors and you actually have to watch how your code is structured and if your syntax is good [it] helps them really be involved with it and not just putting puzzle pieces together because then it just becomes well I just put this here and it worked, it did what it was supposed to do. I don't really know what these blocks do but when I put these four red blocks together it does what I want it to do. I think that works really well in like 1st, 2nd grade but by 5th grade they need to be given the freedom to communicate in the way that we would expect them to communicate which is really text unless the toolkit is scaffolding so they can continue to know what their options are and understand where to go for syntax but I think it'd be frustrating because you can always block something in and click together and well they just don't appreciate and enjoy

what they're doing. So it opens up the flexibility and makes it a more appealing process. Another thing is it makes it feel like they are doing something real. There is a little bit from a block environment where they tell them well I don't think you can really do this and there's this idea that they can do block programming but Python's hard. I heard that million[s] [of] times and it used to drive me nuts! I used to hear teachers say well we did Scratch with them and then tried to get the ones who were really good to try Python but it was very hard and it was like no it was just you were giving them a really boring curriculum that was designed for college kids and no matter how much you simplify it, it was just boring. It wasn't hard, it just needed to be presented in a developmental level for the students.

A: Oh for sure and I think that's one of the biggest issues is at the education level they imagine learning program like it HAS to be this way. Like it has to. We need to have them make a calculator program using methods or something and you know its all super technical and scientific and at this age level it's just not appealing to kids.

G: We're slowly but surely coming to [a point where its] like somebody has a calculus textbook and asking how do we teach this to 7th graders and they're just trying to dumb it down but some of this stuff just doesn't need to be taught and the terminologies [are just] wrong and this sequence of things is just different. So going to back to your question of why loops before conditionals at some point we have to go back and look at developmental reasons like why do we teach them fractions before algebra, why do we teach them to count money in 1st grade like we have over the past 100 years. [Well we] have found a proper sequence of math and science topics and a way to present them at different ages and I think at some point we are going to get there with coding. There is a time when you teach them routines [with] maybe some looping or

matching but you won't be able to teach them functions and variables in 2nd grade, it's just not a thing. It's the same thing in math, there is a reason why we don't teach kids functions and variables in math in 2nd grade [and] it's the same reason why we wouldn't do it with coding. At some point we need to make a coding curriculum that acknowledges where students really are at each grade level not what prior exposure they have. I think we've been through a couple years now and I think that our curriculum is based on a sequence of prior exposure rather than a developmentally appropriate level and [a] sequence is where we need to get to eventually.

A: I think that is very present in your programs, especially when you get to your later assignments. Like if you didn't finish Tim the Wizard you get to Rock, Paper, Scissors and you're stuck!

G: [laughs] Which is the way a general math curriculum is going to look. It's going to look just like that, if you look at a 7th grade math book it creates a certain set of skills and by Chapter 6 if you didn't do chapters 1,2,and 3 your work is really going to suffer because we are going to assume you know things. And yet all of it is presented in a way that makes sense for a kid that [is in this] grade or [is] developmentally. We've seen that they've gone through 6th grade math so we're not dropping something on them out of the blue [its] all a part of something that gets them to calculus [because] at some point but we're not introducing integrals and derivatives in 7th grade because we don't think they can understand it. And I mean they can't, in general, I mean most kids. So we need to get to that point where we are really recognizing [this] in our computer science curriculum but there's no research [so] we are just creating this from blue sky which is what's exciting about it for me. Like god if I had to sit down and think of a new way to teach math I'd lose my mind, a lot of people a lot smarter than me have been working on this

for a long time and the idea that I've worked with middle schoolers for 20 years and if we want to think about how to construct a coding curriculum for middle school kids, I think I can think about that from the standpoint of how do we teach them things? Where are they developmentally? Where are they in science, in math? What are they interested in, what are they working on, how does it tie in? Rather than this is the standard freshmen CS1 curriculum and how do we teach them that, I don't think it'll work that way.

A: Well the good thing is the amount of research of CS education is so plentiful and I'm so happy that it is at the place where it is now so hopefully we'll see a lot more new things that are going to help better schools curriculum.

G: Yeah and I'm excited to see what comes of your research and [be sure to] share with me [because] I've long wanted to get that sort of review of what we're doing. I'd like to understand what things of ours works, what things don't and where we can improve. Whatever you can share with me with data and feedback and thoughts I'd love to hear more about it.

A: Well then hey I'll give you one little bit of information here then. So I gave the kids just last week a little paper quiz on the variable lessons and some of them hadn't finished all of the variable lessons and I at least wanted them to get up to star variables but some of them weren't quite there yet. But what I found was if they at least got 2 or 3 lessons done they understood – you know the things you guys are always testing them on – hey if I create a variable after I try and use it, why does this give me an error? And they were managing to answer the questions pretty well. I think the question that had the least amount right in the

class was the one that I put on there as a gimme. Which was what color do Strings show up and what color do integers show up as?

G: Well good that's all about memorization stuff and only for our platform but I'm glad! I'm glad they are knowing that much more important you gotta make that variable before you can use it. That's awesome, I love it.

A: I'm happy to report! So actually another question here, Codesters may be seen as a MOOC of sorts. Do you think that Codesters can stand alone and students could log in whenever they want and understand the curriculum by themselves?

G: No, and I don't think it's the design of curriculum it's that I think a 10 or 11 year old will never learn that way. I'll tell you in my 20 years of education I've seen that the most important thing to have is an empathetic adult to help guide them through the learning journey. Whether the person helping is an expert is irrelevant. If you have a reasonably intelligent but more importantly empathetic adult in the room then a 10, 11, 12 year old can use our curriculum or any other curriculum. I don't think there will be any product that a 10, 11 year old, in general, again maybe there's certain kids but the general population will be able to learn from independently. I just don't think that is an educationally sound idea I don't think you'll find that it works. So I think MOOCs at a college or professional level might make sense but I don't think a MOOC should be in a middle school classroom. Which is why we made the original decision to partner this with teachers and give teachers the tools and lesson plans and the dashboard because they have the ability to help. You need to have an adult at that age to help them through the learning process. You're just not going to find that kids can do it themselves. I

mean maybe one day we get to the point where the AI is so powerful but you know what we have now in terms of instructional methods like videos and feedback and quizzes, it's just never going to get there. There's just something about a present empathetic adult that is essential to the education of kids.

A: Exactly. I know some people look at this curriculum and think oh it's a MOOC, I can sit them down in front of the computer and have them read what they are supposed to do and then do it but that just doesn't jive well.

G: You gotta be in there to help them do it. And I wouldn't characterize what we have as a MOOC, I would define it as a well scaffolded classroom instructional tool and the scaffolding is really critical because a teacher can only give a certain amount of time when there are 30 kids in a room to each kid. So basically we have to fill in enough structure so one teacher can help 20-30 kids get through so we can get 95% of the structure there but we always assume that the teacher is there. Some of them are very skillful with pairing kids up or ask three then me or other various ways to have kids peer tutor but all the scaffolding there came from our early attempts to create a curriculum for the classroom and recognizing the limiting factors for the teacher and the amount of time they have for each student. We have to give enough scaffolding so the teacher can go to each student and do a bit of teaching. But we can never assume that it will be a full independent curriculum, that's not going to work, we aren't going to replace teachers. That scaffolding is there to try and help a lot of kids at once.

A: Understood. So my next question I have for [you], a few of my students wanted me to ask you and it has to do with diversity. Now what do you think Codesters is trying to do to include

those less prominent groups in computer science. For instance, the girls in the class feels as though it's not really aimed at them and that it's something boys do.

G: So I'll answer with a couple different things. One we very much do partner with girls in computer science, for instance we are a partner of hackergal which right now is running the world's largest female hackathon in Canada. We've partnered with Girls Who Code for years and one of the reasons why we are in so many tier 1 schools is because we've done a lot of work for getting codesters into low income minority communities in places like Brooklyn. So we do actively work on those things. With that said what we've decided, and I think this is a critical thing, is by going directly into the school day and saying this needs to be schoolwide or district wide, every single kid needs to do this. We've tried to avoid the major pitfalls we've seen that has to do with selection bias or who gets to sign up and stay after school and isn't already doing something else. Who [gets to] sign up for that elective class? No no, we need to get every kid doing this and another thing is we get them earlier. So if we get kids doing this in 6th or 7th grade they're less likely to get into that mindset where they decide well I'm not into coding and this isn't for me. And we do find that the critical thing [is] you can't do this in high school you have to do this now when they are developing this self-identity. So getting in there then and showing them that it can be fun and we tried to build our curriculum to be as gender, as race, as everything neutral as we can be. So it isn't targeted at girls but it also isn't targeted towards boys. We try to have a mix of boys and girls as the avatars, a mix of races, we have disabled [avatars]. We have a variety of different students in our curriculum represented in our sprites and we've tried as much as possible to be sort of culturally neutral. We wanted something that can be applicable in Brooklyn but also in Arkansas and we work in both of those places so we

want this to be represented across a spectrum of students. The question has always been, how do we get this to reach kids in every school and I think this approach has really worked. I think for example a lot of program that try and bring in a lot of tech volunteers really limited their geographic skill, when we get in the communities when there are no tech volunteers and no one is willing to go out there. We'd like to make it where you don't have to hire a coding teacher you can use existing teachers because that's the main barrier for most schools because a lot of administration ask do I have someone here who can do this, so we focus on you don't need to have any specially trained instructors, you don't need to bring a tech volunteer in, kids don't have to choose to do this, this is something that they all do. We've wanted this to be a representative sample of all the kids in the country and augmented that with organizations who do reach targeted communities but we wanted something that could reach all 50 million kids all from K-12 as the core question we were trying to answer and really not focusing on just the boys, just the girls, or just the low income students really focusing on generally accessible, a generally applicable curriculum that any school could use regardless of what they have going on.

B: Evaluations used in study

B.1: Pre-Test

Name: _____ Date: _____

Pre – Test

1.) What would be the result of the following equation: $y = \left(\frac{x}{3} + t\right) - 10$ if x = 9 and t = 1?

2.) If x = 10, what would the value of y be if y = x + x*2 – x?

3.) What is 3/4 of 200?

4.) Andrew achieved these scores on these tests in his classes. Which was his best score?

a.) 29/50 for Science

b.) 34/60 for History c.) 44/80 for Math

5.) If a car is traveling at 60 miles per hour, how long would it take for it take for the car to reach a town that is 150 miles away.

a) 2 hours

b.) 2.5 hours

c.) 3 hours

d.) 1 hour

6.) What number added to itself three times is equal to 72?

7.) If every time you got into and got out of a car or bus you got $1. How much money would you earn if all you did on a certain day was leave the house, go to school, and then come back home?

8.) Let's say our friend Jeff is at a fork in a road where he could take one of two paths. The sign for the path on the left says that only people less than 13 and whose names have four letters can go down it. The sign for the right path says that you can only go down it if you are older than 13 and your name is John. If Jeff is 12 years old, which path should he take?

9.) Choose your three favorite ice cream flavors. If you want a cone with two scoops, how many different combinations of flavors could you have?

10.) Given the incomplete sequence of: 1, 2, 5, 8, 1, 2, 5, 8, 1…

What would be the next 3 numbers in the sequence?

B.2: Variable Test

Name: _____ Date: _____

Variable Test

1.) In Codesters, String variables are _____ and integer variables are _____.

a.) Red and Green b.) Blue and Orange

c.) Green and Blue d.) Orange and Blue

2.) If we write the following statement would it work? If not then why not? *"my_var" = 3*

3.) Which of the following would be the correct way to create a String variable called *name* that stores the value "Andy"?

a.) "name" = Andy b.) name = "Andy"

c.) name = Andy d.) name = 4

4.) If we have variables *height* and *width*, which of the following statements would use them to create a Rectangle properly?

a.) codesters.Rectangle(0, 0, "width", "height", "red")

b.) codesters.Rectangle(0 0 width height "red")

c.) codesters.Rectangle(0, 0, w, h, "red")

d.) codesters.Rectangle(0, 0, width, height, "red")

5.) If our code was presented in the following order, would Codesters run it or give us an error? If no then why?

Shawn.move_left(200)

Shawn = codesters.Sprite("person3")

B.3: Loop Test

Name: _____ Date: _____

Loop Test

1.) What is the reason for using loops? What is their main purpose?

2.) How many times will Phil say "Good Morning Class!" if it was in the following loop:

```
for counter in range(5):
    Phil.say("Good Morning Class!")
```

3.) What's wrong with the following loop statement? (two things)

```
for counter in 10
        #Do something
```

4.) What will be the final result of x?

```
x = 3
for counter in range(10)
    #Add 3 to x every time
```

5.) If num is a variable equal to 3, is the following loop valid? Why or why not?

for counter in range(num):

B.4: Conditional Test

Name: _____ Date: _____

Conditional Test

1.) All of the lines included in an if statement which are listed after it are all _____.

a.) deleted b.) added

c.) indented d.) in order

2.) What's wrong with the following if statement? (Think about what's missing)

```
If number == 3
   number += 2
```

3.) What'll happen if the value of name is John?

```
If name == "James":
   Sprite.say("hello!")
If name == "John":
  Sprite.say("goodbye!")
```

4.) Write an if statement that checks if a variable named *hello* is equal to 5.

5.) Let's imagine a rock paper scissors game. Write the if statement that would check the situation where the user chooses rock and make another right under it where the user chooses paper. (Name the variables whatever you'd like and assume rock and paper are Strings!)

B.5: Post-Test

Name: _____ Date: _____

Post – Test

1.) Would the given code run if it was presented in the following order? If not then why?

John.flip_up_down()

John = codesters.Sprite("person3")

2.) In the lesson Robot Design a rectangle was defined like: *sprite = codesters.Rectangle(x, y, width, height, "color")*

Which line of code will create a red Rectangle with a height of 5 and a width of 3 at the coordinate (7, 10)

a.) sprite = codesters.Rectangle(7 10 3 5 "red")

b.) sprite = codesters.Rectangle(7, 10, 3, 5, "red")

c.) sprite = codesters.Rectangle(7,10,3,5,red)

d.) sprite = codesters.Rectangle(3,5,7,10,"red")

3.) Given four variables: *age = 10, school = "Central City", name = "Michelle"* and *color = "green"*

Which of these variables would be a String? Which would be an int?

4.) What would be the result of my_num if it is equal to the following equation:

*my_num = (num1*num2)/final*

Where *num1 = 10, num2 = 5,* and *final = 25*

5.) Which of these represents an equation for the Area of a Triangle (A = ½bh) using code where your variable for b is *base* and your variable for h is *height*?

a.) A = (1/2)*b*h

c.) A = b*h

b.) A = (1/2)bh

d.) A = (1/2)b*h

6.) Let's say a variable *number* is equal to (*x/2)*3*, what would its value be if x = 10?

a.) 30

b.) 10

c.) 15

d.) 7

7.) How many times does the sprite named James say Hello Everybody!

```
for counter in range(10):
    James.say("Hello Everybody!")
```

a.) 1

b.) None

c.) 7

d.) 10

8.) What would be the final value of x?

```
x = 10
for counter in range(5):
    x = x + 3
```

a.) 3

b.) 25

c.) 10

d.) 5

9.) How many times would this for loop run if number = 12

for counter in number:

10.) What will Tim say if the *rand_num* is equal to 2?

```
If rand_num == 1:
    Tim.say("Hey!")
If rand_num == 2:
   Tim.say("Hi!")
```

11.) Let's say that Jim's age is represented by the variable *age*. Based on this, which path should Phil take if he is 13 years old?

```
If age > 10:
    #Take path on the left
If age < 10:
   # Take path on the right
```

12.) Which of the following if statements checks to see if a variable *num* is equal to 7?

a.) *if num == 5:*

b.) *if num == 7*

c.) *if if if == 5*

d.) *if num == 7:*