Journal of Information Technology Education: Innovations in Practice Volume 15, 2016

Cite as: Du, J., Wimmer, H., & Rada, R. (2016). "Hour of Code": Can it change students' attitudes toward programming? *Journal of Information Technology Education: Innovations in Practice, 15,* 52-73. Retrieved from http://www.jite.org/documents/Vol15/JITEv15IIPp053-073Du1950.pdf

"Hour of Code": Can It Change Students' Attitudes toward Programming?

Jie Du
Grand Valley State University, Allendale, MI, United States
dujie@gvsu.edu

Hayden Wimmer
Georgia Southern University, Statesboro, GA, United States
hwimmer@georgiasouthern.edu

Roy Rada
University of Maryland, Baltimore County, Baltimore, MD, United States
rada@umbc.edu

Abstract

The Hour of Code is a one-hour introduction to computer science organized by Code.org, a non-profit dedicated to expanding participation in computer science. This study investigated the impact of the Hour of Code on students' attitudes towards computer programming and their knowledge of programming. A sample of undergraduate students from two universities was selected to participate. Participants completed an Hour of Code tutorial as part of an undergraduate course. An electronic questionnaire was implemented in a pre-survey and post-survey format to gauge the change in student attitudes toward programming and their programming ability. The findings indicated the positive impact of the Hour of Code tutorial on students' attitude toward programming. However, the students' programming skills did not significantly change. The authors suggest that a deeper alignment of marketing, teaching, and content would help sustain the type of initiative exemplified by the Hour of Code.

Keywords: computer science education, advocacy, Hour of Code, Code.org, online tutorials, introductory computer programming, survey.

Introduction

Enhancing students' attitudes toward programming has long been a hot topic for educators. One website, Code.org, created an enormous interest in its Hour of Code initiative. Code.org is a non-profit organization that was founded in 2012 and whose vision is every student in every school should have the opportunity to learn computer science (Guynn, 2013). The organization operates largely as a virtual organization associated with its web site www.code.org. One of Code.org's most famous initiatives is called the Hour of Code which encourages students to complete short programming tutorials.

Editor: Bronwyn Hegarty
Submitted: May 12, 2015; Revised: September 25, December 20, 2015, March 4, 2016;
Accepted: March 23, 2016

Code.org developed a catalog of tutorials that are suitable for an introduction to programming, and this paper refers to these as either Code.org or Hour of Code tutorials. As of March 2015, over one hundred million students have done the Hour of Code. Assessing student experience with some of the tutorials is the primary goal of this paper.

A review of the tools available to support the teaching of programming concludes that more needs to be done to help students learn to code (Daly, 2009). Even though computer programming is deemed important, there is a lack of student interest in programming. This study aimed to identify the attitude change of students towards programming after they finished an hour long tutorial at Code.org, investigate whether their skills changed, and was designed to provide insights for the dissemination of computer science education.

The Code.org's Hour of Code is delivered annually during Computer Science Education Week. The authors reported on a study conducted across two universities in the United States. A sample of undergraduate students was asked to do a one hour tutorial at Code.org as part of their course. An electronic questionnaire was implemented to survey these students before and after they completed the online tutorial regarding their attitudes toward programming and their skills for coding.

The remainder of this paper is organized as follows. First, a brief review of the literature and background information on Code.org and other approaches to teaching programming visually is described. Next, the methodology is presented followed by results. Subsequently, a discussion of the results occurs with a conclusion completing the work.

Literature Review

STEM, or science technology engineering and math, education is of national interest with an ever increasing amount of federal grant funding supporting developing STEM education. Primary education is overwhelmingly lacking in technology subjects, specifically (Sanders, 2008). Industry alliances, such as Microsoft, VMWare, and EMC, have joined with universities and, in some cases, primary and trade schools to offer technology to students. Computer programming is one critical aspect to STEM education that poses especially difficult challenges.

Computer programming is not a task easily conquered by a novice. This results in lower student retention and difficulties passing programming courses. Student attitudes toward programming have been studied for decades (Koohang, 1989) and shown to cause anxiety (Raub, 1981). In fact, the phenomenon is so commonplace that researchers have collaborated on developing a standardized survey (Elliott Tew, Dorn, & Schneider, 2012; Kay, 1994). Besides student grades, retention, attitudes, and anxiety, studies have indicated gender differences in introductory programming courses (Rubio, Romero-Zaliz, Mañoso, & Angel, 2015). Java is one of the most common languages employed in introductory programming courses; however, Java's peculiarities make it difficult to learn (Pendergast, 2006). The academic literature is teeming with studies on methods to facilitate programming education. Languages have been shown to have an impact on comprehension. For example, Nikula, Sajaniemi, Tedre, and Wray (2007) demonstrated that languages with a higher level of abstraction, namely Python, improved student retention and comprehension. Similarly, visual tools for programming can be employed to aid novice programmers overcome the aforementioned difficulties (Moor & Deek, 2006). Following the visual programming strategy, Lee, Pradhan, and Dalgarno (2008) argued that visual tools facilitate a novice programmer to develop and manipulate mental models and schemas.

Gaming strategies are considered more enjoyable than traditional training environments (Venkatesh, 1999). Goel and Kathuria (2010) demonstrated that having students work on sub problems and combine the sub problems to solve a larger problem is effective in improving the quality, efficiency, and teamwork of introductory programming students. Games involve working on smaller tasks to achieve a goal. For example, many video games have levels and once all lev-

els have been successfully completed the player wins the game. IBM's Robocode is an environment that improved users programming skills while deemed an enjoyable experience (Long, 2007). Code.org seeks to intersect the visual approach with gaming strategies.

What Is Code.org?

Code.org is a non-profit foundation headquartered in Seattle, Washington. According to its charter, Code.org wants to help make computer science education available to all United States of America (USA) students in all K-12 public schools by 2020. Its charter further says:

> Code.org is ... leveraging years of foundational effort by the National Science Foundation ....Code.org plans to ...: 1) Educate; 2) Advocate; and 3) Celebrate .... The 'Educate' strategy consists of bringing computer science to schools by developing our own curriculum, vetting and recommending additional third-party-designed curriculum, and training teachers to implement and use these curricula in the classroom. The 'Advocate' lever involves changing the rules in states that currently don't recognize CS as satisfying graduation credits in math and/or science.... Finally, the 'Celebrate' piece involves inspiring students, parents, and schools to want to participate through high-level marketing via videos, events and celebrity endorsements. (GuideStar, 2014)

As of 2013, the 'Educate Initiative' has a budget of $3.3 million that was to provide 1) $10,000 per teacher for training and 2) the development of an open-source curriculum in the programming language Block. The 'Advocate Initiative' has a budget of $900,000 for advocating computer science education. As of 2013, in 33 of 50 USA states 'computer science' does not count toward K-12 math or science requirements. Also, if a student studied computer science this subject failed to meet graduation requirements. In 2013, Code.org helped change this policy in Washington State and aims to do so in the 35 other states. The 'Celebrate Initiative' has a budget of $900,000 and aims to use marketing, celebrities, and events to motivate students, parents, and teachers to support computer science education. This paper focuses on investigating the 'Celebrate Initiative' by studying students' attitudes toward programming after they complete an Hour of Code tutorial. Code.org said that the metric of success for the 'Celebrate Initiative' was the number of students who attempted to learn the Hour of Code online. The Hour of Code is a one-hour introduction to computer science and organized by Code.org. The goal of the Hour of Code is to demystify code and show that anybody can learn the basics. Participation in computer science week in 2014 was represented by 180 countries and 76,000 students in the classroom (Wilson, 2015). Code.org has extended its reach in computer science education by providing curriculum guides for AP computer science (Franke & Osborne, 2015) and is being deployed in K-5 and K-12 curriculum (Apone et al., 2015).

Approaches to Teaching Programming Visually

Code.org employs a visual approach to teaching computer programming. Visual approaches have been shown to enhance program comprehension (Rajala, Laakso, Kaila, & Salakoski, 2008) and may be paired with goals and plans (Hu, Winikoff, & Cranefield, 2012) or screencasts (Powell, 2015). Other visual programming tools have been employed in computer science education. One such tool is Alice (Dann, Cooper, & Pausch, 2011). Alice is a virtual environment where students use visual programming in the form of puzzle pieces to construct code that causes actors in a virtual environment to perform tasks (i.e., a rabbit hopping). Alice was employed in a general education programming course for students who did not require the in-depth knowledge of a computer science course (Ali & Smith, 2014). Alice was shown to be an effective alternative to standard programming languages in teaching programming to undergraduate computer science students (Sykes, 2007) as Alice's approach addresses many of the issues plaguing programming education (Ali & Smith, 2014).

Like Alice, Scratch is a visual tool for teaching computer programming, primarily for ages 8-16 (J. Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). Scratch involves tasks such as drawing and animating characters, creating stories, and games (Resnick et al., 2009). Scratch has been deployed as a tool for teaching programming to inner-city youth (J. H. Maloney, Peppler, Kafai, Resnick, & Rusk, 2008). Scratch has proven its worth in preparing at-risk computer science students for further computer science courses (Rizvi, Humphries, Major, Jones, & Lauzun, 2011). Additionally, universities have used Scratch as a first computer programming course ultimately transitioning students to Java or other languages (Malan & Leitner, 2007).

Similar to Alice and Scratch, MIT App Inventor is a visual programming tool where students build code by assembling blocks or pieces of a puzzle (Wolber, Abelson, Spertus, & Looney, 2011). App Inventor creates mobile applications for android devices. As part of a Google pilot program, App Inventor was deployed at the University of San Francisco and gained interest since students could create applications for their devices (Wolber, 2011). App Inventor was also deployed with a template-based approach to mobile programming for electrical engineering students and, judged by surveys, proved successful (Akopian, Melkonyan, Golgani, Yuen, & Saygin, 2013). Demonstrating its flexibility to teach introductory programming concepts, App Inventor has been deployed at summer camps for high school students in Georgia with mixed results (Roy, 2012).

Around the World

Programming education is widely recognized as important to a country, and research is being conducted on different approaches that may be necessary in different cultural contexts (Apiola & Tedre, 2012). At a workshop in Saudi Arabia on computer science education, the Code.org initiative was introduced (Rada, 2013). Another speaker at the same workshop, Sami Al-Wakeel, explained that Saudi students had already been introduced to computing and thus an initiative such as Code.org's Hour of Code might be less relevant in Saudi Arabia than in the USA (Al-Wakeel, 2013). Al-Wakeel served as leader of the National Computer Education Committee of the Saudi Arabian Ministry of Education from 1996 to 2006 and described how Saudi Arabia decided that computer science education for all students at all levels was important to its economic development (Al-Wakeel, 2001). Accordingly, the government installed networked computer labs in all secondary schools and most elementary schools. Additionally, the Saudi government created new teacher training departments in teacher education colleges so that ample computer science teachers were educated. In such a system, the Code.org initiative is not as relevant, as all students have already been exposed to computer science. Singapore has one of the world's most advanced educational systems. Unlike the Saudi approach and the Code.org approach, the Singaporean goal is not that everyone learns computer science but that everyone uses computers to improve learning and a large fraction (but not all) of the population participates in the program.

As computers become more pervasive in society the need for skills in computer programming increases. Research demonstrates that learning and teaching computer programming is challenging and student retention proves difficult. Code.org, with support from the National Science Foundation, has emerged as an advocate for computer science education with programming tutorials as well as international outreach. With the emergence of Code.org's Hour of Code initiative, research is required to understand how these tutorials can improve students' attitudes and comprehension of computer programming. Two hypotheses were examined in this research.

1. The Hour of Code tutorials at Code.org would significantly enhance students' attitudes toward programming.
2. The Hour of Code tutorials at Code.org would significantly improve students' skill for coding.

Methodology

Students were asked to undertake one tutorial, as part of its Hour of Code initiative. The procedure, data processing, and analysis are described in the forthcoming sections.

Procedure

Undergraduate students, especially freshmen and freshwomen, were randomly selected from required core course sections in an effort to capture a diverse group of students within the sample. Undergraduate students from two universities participated. In particular, one hundred and sixteen students with a spectrum of majors including business, accounting, criminal justice, allied health sciences, geography, hospitality tourism management, and psychology agreed to take part in this study. Among them, 44 students were from one university (call it A) and 72 students from the other university (call it B).

Data collection

The methods for collecting data involved three steps.

1. Pre-survey.
2. Online tutorial.
3. Post-survey.

Pre and post surveys were used to study the change in students' attitudes and skills based on the work by Bouhnik and Giat (2009). An electronic pre and post-test Likert-scale questionnaire was implemented to survey the participants about their attitudes toward programming (see the Appendix). Participants' responses to the pre and post surveys were matched using a PIN number randomly created for each participant. The electronic questionnaire included a variety of questions relating to programming experience and attitudes toward programming. Background information such as participants' programming experience was captured first. Participants were asked whether they had ever taken any programming courses. Next, participants were asked the number of years of programming experience.

The next set of questions was targeted on understanding participants' attitudes toward programming (see Table 1). Among the four questions relating to students' attitudes toward programming, three (Q1, Q3, and Q4) used a five-point Likert scale: "Strongly disagree", "Disagree", "Neutral", "Agree", and "Strongly agree". Question 2 (Q2) used a four-point Liker scale: "Very likely", "Moderately likely", "Slightly likely" and "Not at all likely". Another set of questions was related to programming knowledge. Three questions were asked to test participants' understanding of three basic programming concepts: sequence, if-then, and loop. For each programming question, four options were provided and students were asked to select the best answer. At the end of the questionnaire, participants were asked to provide additional comments regarding programming. In summary, the pre-survey contained eight questions covering the topics mentioned above. The post survey contained the same eight questions that appeared in the pre-survey plus two new questions (Q3 and Q4) with the assumption that the students' attitudes toward programming and their coding skills would change as a result of taking the tutorial.

Table 1. The four questions relating to attitudes towards programming.

Q1	To what extent do you agree or disagree with the following statement: Everybody in this country should learn how to program a computer because it teaches you how to think.
Q2	How likely are you to take a programming course?
Q3	Did you enjoy the tutorial provided by Code.org?
Q4	Completing the tutorial changed your attitude towards programming how?

Coding tutorial used in the study

Given that most of the participants were freshmen or freshwomen and had very limited programming experience, the tutorial "write your first computer program" from the category of "Tutorial for Beginners" was selected to use in this study. The participants were asked to complete the tutorial in class. This tutorial invited the student to work through 20 progressively more complex mazes to get an angry bird to reach a pig (see http://learn.code.org/hoc/1). The student could cause the bird to take one step forward, left, or right using the programming language called Block. The only other instructions available to the student were to repeat an instruction or to make an if-then decision. The student began with the simple maze in which the pig is two steps in front of the bird. If the student solved the maze, then the student was given positive feedback and presented a more challenging maze. For instance, the 8th maze involves 'do loops' (see Figure 1). The student was given feedback after each attempt. After a few successful mazes had been solved, a video appeared to introduce the next sequence of mazes. The videos were presented by famous people, including one by Bill Gates and another by Mark Zuckerberg. For instance, after the 'do loops' sequence of mazes, the student was introduced to the 'if instruction' by Bill Gates (see Figure 2). When the student came across difficulties, he or she was encouraged to first ask the classmates for help. The Hour of Code emphasized the value of having students turn to one another for help.

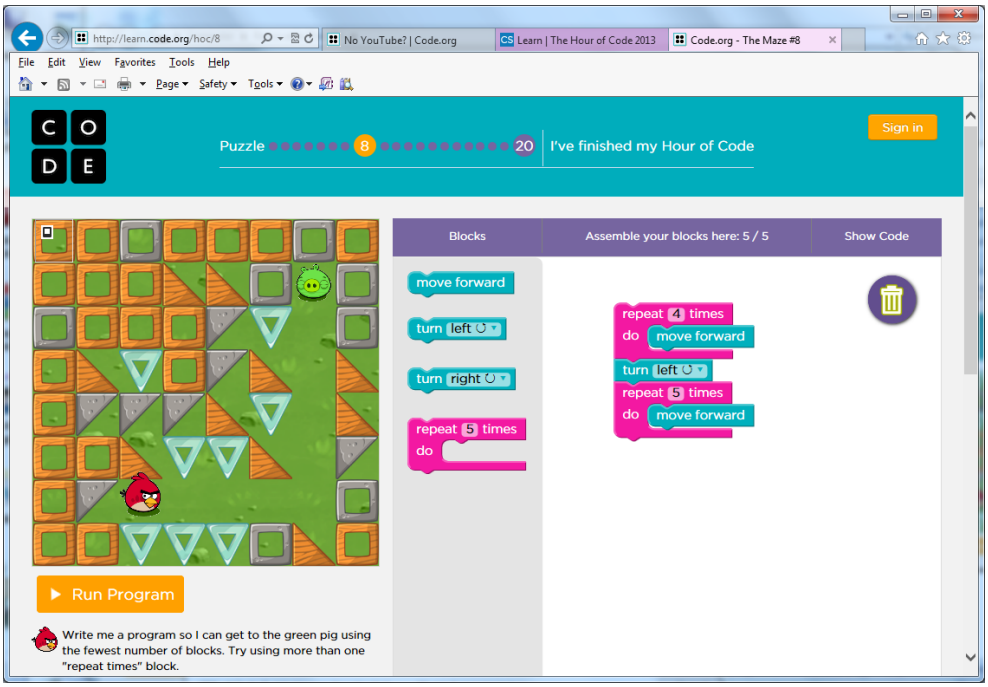


Figure 1. "Do Loops"

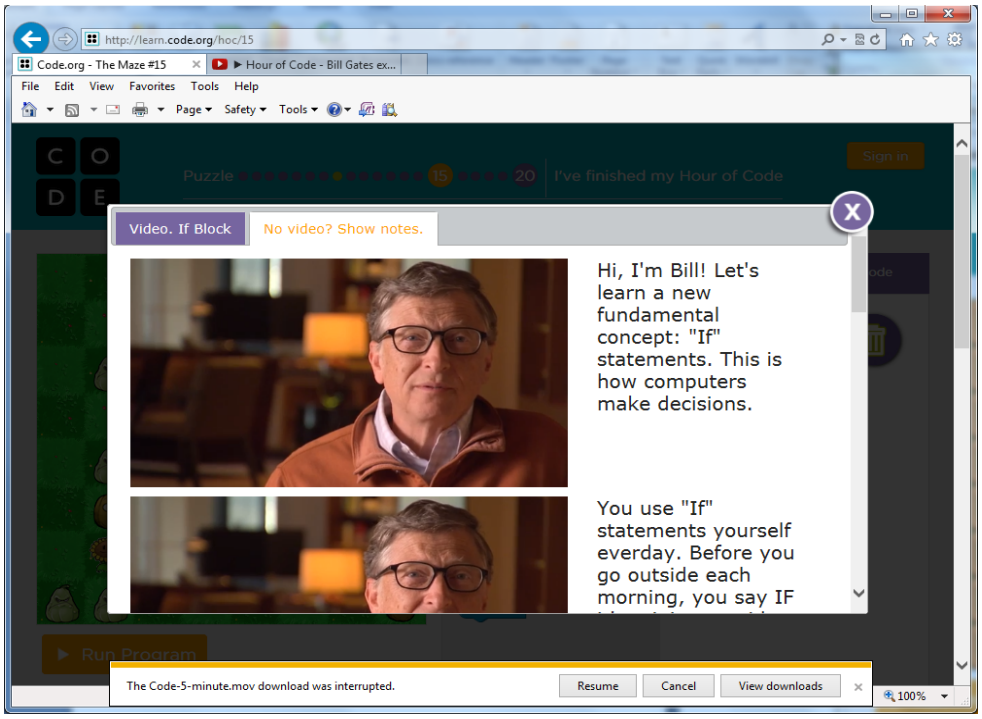


Figure 2. Bill Gates Video

Data Processing

The participants were asked to take the electronic questionnaire before and after they completed the tutorial, "write your first computer program". All data were anonymized so students could not be individually identified. Student responses were collected automatically by a secured third party online survey platform (http://www.qualtrics.com/).

The data were processed differently for the questions relating to attitudes toward programming and programming experience. In order to compare the change of students' attitudes toward programming, for the questions relating to attitudes toward programming, the participants' response to the Likert-scale questions was processed into a binary number, 1 or 0 depending on the positive or negative response.

The responses, r captured by Qualtrics.com were recorded as 1 to 5, corresponding to strongly disagree to strongly agree, respectively. Next, the r was processed into a binary value, x (see Eq. 1). Only the positive responses (agree and strongly agree) were recorded as one. Other responses were recorded as zero.

$$x_{ij} = \begin{cases} 0 \; when \; r_{ij} < 4 \\ 1 \; when \; r_{ij} \geq 4 \end{cases} \qquad \text{(Eq.1)}$$

Where r_{ij} is Student i's original response to Question j and x_{ij} is the processed Student i's response for Question j.

For the questions relating to programming experience, the participants' response was processed into a binary number, 1 or 0 depending on whether it matches the correct answer. Four options were provide for each programming question. Students were asked to select one that best answered the question.

In this case, the responses, *r* captured by Qualtrics.com were recorded as 1 to 4, corresponding to *a* to *d*, respectively. Next, the *r* was processed into a binary value, *x* (see Eq. 2) depending on whether it matched the correct answer.

$$x_{ij} = \begin{cases} 0 \; when \; r_{ij} \neq correct \; answer \\ 1 \; when \; r_{ij} = correct \; answer \end{cases} \qquad \text{(Eq.2)}$$

Where r_{ij} is Student *i*'s original response to Question *j* and x_{ij} is the processed Student *i*'s response for Question *j*.

For example, a question related to programming concepts was:

In the following pseudocode, what is printed?

```
a = 1
b = 2
c = a
a = b
b = c
print a, b
```

Choice of responses.

a. nothing
b. 1 2
c. 2 1
d. None of the above

The correct answer is *c*. The response was coded as 0 when the participant chose the incorrect answer and 1 when he or she selected the correct answer, *c*.

Analysis

The hypotheses suggest that the Hour of Code tutorials at Code.org would significantly enhance student's attitudes toward programming as well as their coding skills. In order to test the hypotheses, students' responses from both pre and post survey were compared. The comparison analysis contained two steps. First, the frequency of different responses for the four attitude questions was compared. One can learn basic information about the changes from those numbers, such as increased (or decreased) positive (or negative) responses for certain questions. Accuracy is defined as the percentage of the programming questions that were correctly answered and used to evaluate students' performance on the three programming questions. Accuracies between two universities were compared.

It should be noted that frequencies cannot be used to tell whether the responses from pre and post survey were statistically different. A statistical test is necessary for measuring significance. Therefore, the second step was to conduct a two-tailed, paired-sample *t*-test to examine whether the difference between pre and post survey responses was significant. The *t* value and *p* value were used to evaluate the difference.

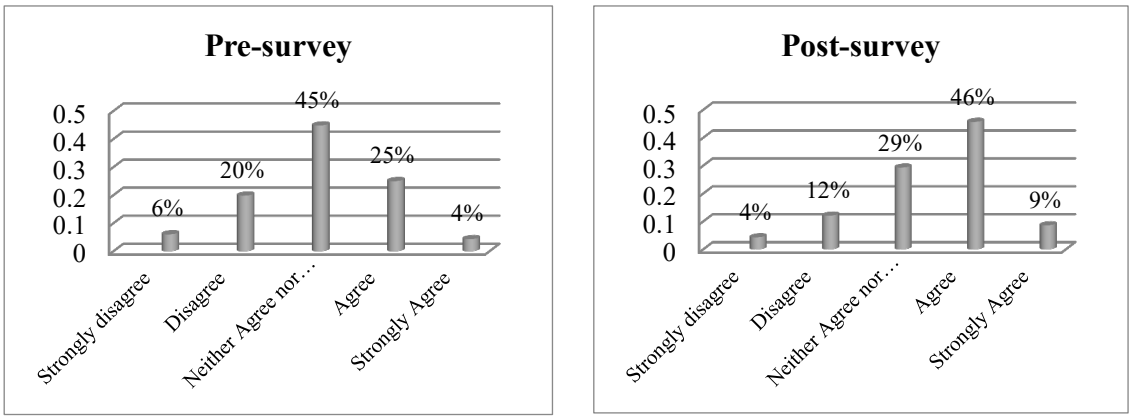
Results

Students were asked to enter the same PIN randomly created for each participant when they took the pre and post-survey so that their responses in the pre and post survey could be matched. The mismatch happened at two situations 1) when two students completed the pre-survey but not the post survey, and 2) when one student did not enter a PIN in both surveys. Accordingly these three responses were dropped from the dataset. As a result, the data set contains 116 students' respons-

es. All comparisons were conducted within one sample of 116 participants. Of the 116 students participating in this study, the majority (72%) had never taken any computer programming courses. The average number of years of programming experience was one year. This information confirmed the assumption that the tutorial "write your first computer program" was appropriate to use in this study due to participants' very limited programming experience.

Awareness of Importance of Programming

In the pre-survey, 29% of the participants believed that everyone in the country should learn how to program a computer compared to 26% who did not while 45% remained neutral. By contrast, in the post-survey, the majority (54%) of the participants indicated that they agreed with the statement, 29% were neutral, and 16% disagreed. Figure 3 shows the distribution of the different responses. It shows that more students have positive attitudes toward programming after completing an Hour of Code tutorial.



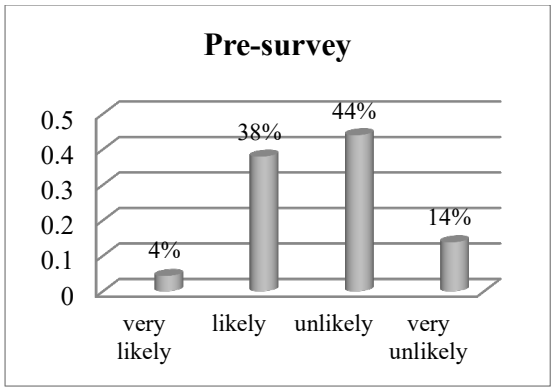
Pre-Survey (n=116)

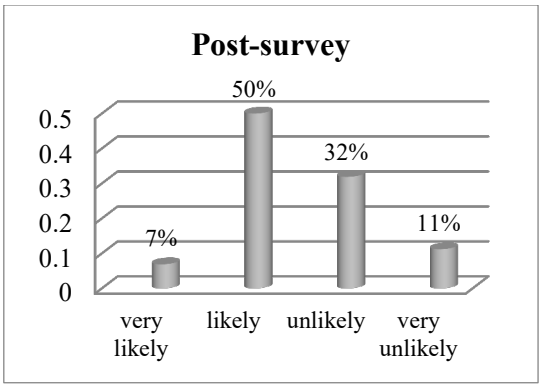
Post-Survey (n=116)

Figure 3. Students' responses to whether everyone in USA should learn to program

There was a significant difference in students' attitudes towards learning programming, before the tutorial (M = 0.29, SD = 0.21) and afterwards (M = 0.54, SD = 0.25); t (115) = -5.92, p = < 0.001. These results suggest that the tutorial changed participants' attitudes toward learning programming. After taking the tutorial, students were more positive about learning programming.

In the pre-survey, the majority (58%) of participants considered it unlikely that they would take a programming course compared to 42% who did. In the post-survey, the reversal occurred with 57% indicating they would do so. The difference between the pre and post survey responses is shown in Figure 4. Students are more likely to take a programming course after completing an Hour of Code tutorial.





Pre-Survey (n=116)

Post-Survey (n=116)

Figure 4. Students' responses to how likely they are to take a programming course

There was a significant difference in students' responses, before the tutorial (M = 0.42, SD = 0.25) and afterwards (M = 0.57, SD = 0.25); t (115) = -3.93, p = < 0.001. These results suggest that the tutorial changed participants' attitudes toward learning programming. After taking the tutorial, students are more likely to take a programming course.

The third and fourth questions were asked only in the post survey. The majority of participants (79%) enjoyed the tutorial provided by Code.org compared to 8% who did not (see Figure 5). When asked about how the tutorial changed their attitude towards programming, the majority of participants (60 %) answered "strongly positive" or "positive" (see Figure 6). Interestingly, over a third (35%) were non-committal. It is possible that working on the tutorial for an hour was inadequate for those participants to detect changes in their attitudes toward programming. One would expect to see an increase in the positive feedback if participants were to complete a series of tutorials. Both Figure 5 and 6 illustrate the distribution of the responses after students completed the Hour of Code tutorial and show that students have more positive attitudes toward programming after completing an Hour of Code tutorial.

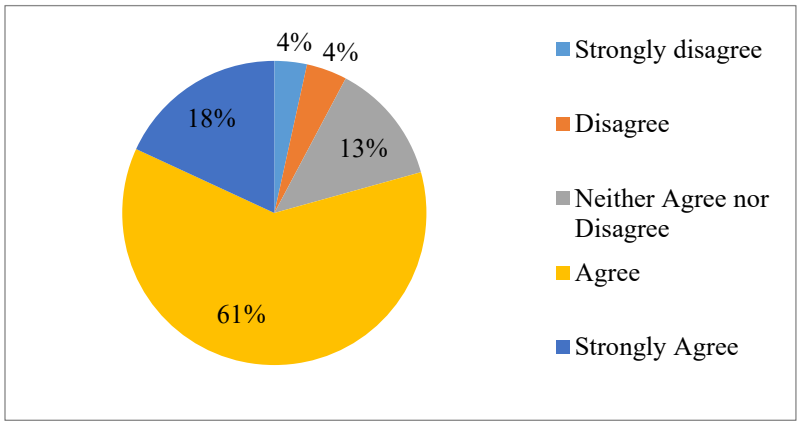


Figure 5. Students' responses to whether they enjoyed the tutorial provided by Code.org (n=116)

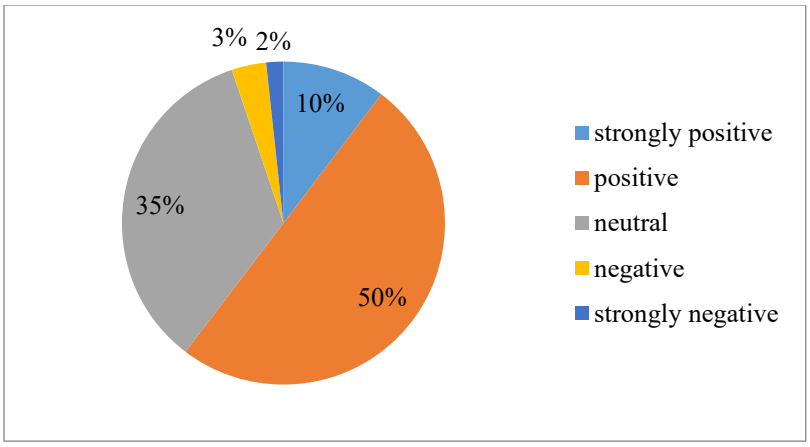


Figure 6. Students' responses to how completing the tutorial changed their attitude toward programming (n=116)

Of the 116 participants, a small percentage, 14% provided additional comments regarding programming. In the pre-survey, one common, repeated comment is "I have no clue what I'm doing", which is expected given the participants' limited programming experience. By contrast, participants' comments were much more positive in the post-survey. The selected quotes from participants' comments represent four themes – fun, interesting, important, and great – that emerged and illustrate that participants appreciated the Hour of Code tutorial.

> **Fun:** *"I thought it would be hard but it was actually fun"*
> **Interesting:** *"It is a very interesting language and although it is complex it can be learned by virtually anybody! It would just take time just like anything else!"*
> **Important:** *"Became much more interested with programming and realized how important it truly is"*
> **Great:** *"Great way to learn! It was a fun, interactive way to learn how to "code"."*

Skills for Coding

Three questions related to programming concepts, loop, if, and sequence, were asked before and after participants completed the tutorial. Questions related to comprehension were similar to those employed by Ford and Venema (2010); however, pseudo-code was employed in lieu of Java or C++. The difference on accuracy was tested by using a two-tailed, paired-sample *t*-test. These differences were not statistically significant. The detailed results are shown in Table 2.

Table 2. T-test statistics comparing skills for coding before and after the tutorial

Questions		n	mean	SD	t	p
Loop	Before taking the tutorial	116	0.41	0.24		
	After taking the tutorial	116	0.47	0.25		
					-1.35	0.179
If	Before taking the tutorial	116	0.72	0.2		
	After taking the tutorial	116	0.66	0.23		
					1.47	0.1449
Sequence	Before taking the tutorial	116	0.19	0.16		
	After taking the tutorial	116	0.18	0.15		
					0.26	0.7975

Completing an Hour of Code tutorial did not make a difference in the accuracy of the three programming skills questions. One possible reason for that is the code in the survey is more traditional and does not closely mimic the code on "the write your first computer program" at Code.org. Another possible reason is that using Hour of Code alone was not effective in teaching students how to program. The Hour of Code was not designed to replace traditional computer classes but could be incorporated into computer science curricula as another dimension of computer science education.

Of the 116 participants, 44 were from University A and 72 from University B. It is interesting to notice that the accuracy of participants varied between the institutions. For the loop question, there was no significant difference for the participants from University A, before the tutorial ($M = 0.48$, $SD = 0.26$) and afterwards ($M = 0.43$, $SD = 0.25$); $t(43) = 0.53$, $p = 0.60$. Therefore, completing an Hour of Code tutorial did not make a difference in the accuracy for the participants from University A. By contrast, there was a significant difference on the accuracy for the participants from University B, before the tutorial ($M = 0.36$, $SD = 0.23$) and afterwards ($M = 0.49$, $SD = 0.25$); $t(71) = -2.59$, $p = 0.01$. Hence, the accuracy was significantly increased for the participants from University B ($p < 0.05$). This increased accuracy demonstrated enhanced understanding of the loop concept for participants who completed the Hour of Code tutorial. The student populations from both universities consisted of first year undergraduate business majors; however, the structure of the courses was slightly different. University A's course covered a more diverse range of computing topics such as networking, I/O, security, and communications whereas University B was focused solely on business and office applications. Since students from University B had more depth in applications they would have more background to understand looping concepts.

For the IF question, no significant difference was detected for the participants from University A, before the tutorial ($M = 0.66$, $SD = 0.23$) and afterwards ($M = 0.68$, $SD = 0.22$); $t(43) = -0.27$, $p = 0.79$. Therefore, completing an Hour of Code tutorial did not enhance the students' understanding of the IF concept from University A. However, there was a significant difference for the participants from University B, before the tutorial ($M = 0.76$, $SD = 0.18$) compared to afterwards ($M = 0.64$, $SD = 0.23$); $t(71) = 2.24$, $p = 0.03$. Thus accuracy, post-survey, was significantly decreased for the participants from University B ($p < 0.05$). Students at University B had reviewed 'if' statements in Microsoft Excel so it is possible transitioning an Excel formula to code caused confusion. It seems that the stylized, visual instructions in the Hour of Code tutorials confused the participants when they were later tested on a traditional format what was usually taught in introductory computer science courses.

In summary, the findings show that completing the Hour of Code tutorials positively impacted the students' attitudes toward programming. However, changes in understanding and accuracy depended on the type of coding questions and the university group. Introducing the Hour of Code tutorials into classrooms sheds some insight into the future development of computer science education. However, sometimes the students were unclear on the purpose of the tutorial, or more generally, the purpose of computing, as manifested by this comment from one student, "*It was fun, yet it didn't actually teach me code. It taught me it a basic coding concept, but I figure it's a first step*."

The primary limitation of this study was the small sample size. Due to the exploratory nature of this study, the convenience sample provided a good foundation for exploring the impact of the Hour of Code tutorials on students' attitudes toward programming as well as their skill for coding. Future work should consider incorporation of more participants from a variety of locations. Another limitation is the inconsistency of Likert scale in the survey questions. One question uses four-point Likert scale instead of the five-point scale. Although it should not cause any bias since the 'distance' between each successive item category is equivalent, it is better to use the same

Likert scale for all questions. In addition, the questions (have you ever taken any programming courses and what's your experience with programming) should not be asked twice (in the pre and post surveys) when the same group was surveyed. These limitations will be addressed in our future research.

Discussion

In this section, the authors first examine how the findings from the study helped to answer the hypotheses. Next, the authors suggest several ways to further improve Hour of Code.

The change of students' attitudes toward programming following the tutorial was statistically significant, and it demonstrated an increased awareness of the importance of programming. Therefore the first hypothesis was proven, and Hour of Code tutorials successfully inspired students toward learning the skills of computer programming.

However, the second hypothesis was not proven as no significant difference was found in the acquisition of skills for coding following the tutorial. The comment from a student in the post-survey highlighted that although the tutorial was fun it was not useful for teaching actual code.

The results shed light for future work. The question format used to test participants' understanding of basic programming concepts needs to mimic the ones adopted by Hour of Code. The Hour of Code utilizes a gaming strategy and emphasizes interactive learning. However the current surveys use multiple choice questions. Using a more interactive question format in a gaming context would help students apply the concepts to solve a problem. This confirms the study of Rajala et al. (2008) where visual approaches aid in program comprehension. Similarly, visual approaches such as Alice have been an effective alternative to standard approaches to programming education (Sykes, 2007) and Scratch is used to transition students to other languages such as Java (Malan & Leitner, 2007). Furthermore, gaming strategies are considered more enjoyable than traditional environments (Venkatesh, 1999) and dividing problems into sub-problems are effective in improving introductory programming students (Goel & Kathuria, 2010). The success of Code.org is likely due to the aforementioned reasons. Code.org follows a visual approach and divides the larger problem into sub-problems as a series of steps. This is all framed within a game context similar to the popular Angry Birds.

The Hour of Code is remarkable for its ability to capture the attention of the mass media and has, of necessity, needed to simplify its pitch to succeed in the mass media. The Hour of Code is not ultimately teaching coding for an hour. Hour of Code is designed to take advantage of information technology to address the challenge of getting people to recognize a problem and be able to convert it into an algorithm. The Hour of Code aims to get students engaged in a tutorial highlighting how a problem can have a solution, expressed as an algorithm which can be translated into code. This code can be run on a computer to solve the problem. Therefore, to help people appreciate the importance of this process, Hour of Code tutorials have a focus on problems that students can readily grasp and on tutorials that are fun to perform. Of course, the purpose is not entertainment or fun but teaching and learning. If teachers are to use the Hour of Code in their classrooms, they need to appreciate the learning objectives of the Hour of Code and integrate the tutorials appropriately into their teaching.

Ways to Further Improve Hour of Code

People are increasingly adept at using simple computing devices through a kind of visual programming which may be a different ability than the needs of classically trained computer scientists. What does that mean for the initiative of Hour of Code? Several opportunities for improvements are discussed next.

Motivation and Synchronization

A student's motivation is crucial in determining whether that student finds a particular resource helpful (Chen & Rada, 1996). Motivational factors might include:

- obtaining credentials toward a certificate or degree,
- securing jobs, and
- the potential for peer support.

In education, if a student wants a diploma, the school generally determines the sequence of 'tutorials' the student must master to earn the diploma. However, since the purpose of the diploma may be partly to help the student get a job, a student's progress in the coding tutorials could be connected to tasks and rewards relevant to finding employment. A simple example of a programming-type task that small businesses might find helpful could be developing a web site to market the business. A student who lives in the same neighborhood could learn about the business's marketing requirements. That same student could learn how to develop a web site through online tutorials. A different example might be tutorials for developing applications for smartphones. Students could sell their applications through app stores.

Students may be intimidated when they are asked to learn programming. This phenomenon was possibly caused by the non-interactive traditional training environments (Cheng, Jayasuriya, & Lim, 2010). The Code.org adopts a gaming strategy and tries to provide more interactive tutorials to students. The vision of Code.org is that students have the opportunity to learn aspects of computer science. An effective way to motivate students is to keep them open-minded toward programming and encourage them to try new things. Students appreciate that effort. Just as one student commented when he completed an Hour of Code tutorial, "*I thought it would be hard but it was actually fun*."

The guides at Code.org for delivering the Hour of Code emphasize the value of having students turn to one another for help. This peer-to-peer learning can be valuable for multiple reasons (Hanks, Fitzgeral, McCauley, Murphy, & Zander, 2011). The benefits of pair programming include increased success rates in introductory courses, higher student confidence in solutions, and improvement in learning outcomes. Systems have been proposed for Massively Multiplayer Online Role-Playing Games for teaching introductory computer programming (Malliarakis, Satratzemi, & Xinogalos, 2013).

As an advocacy organization, Code.org succeeded in marketing online computer programming tutorials. The Hour of Code, as a global movement, is reaching tens of millions of students in over 180 countries. The organizations providing the tutorials and those providing the students were, however, not always synchronized. Perhaps the marketing schedule was overly ambitious? From the authors' perspective, the incentives for alignment of marketing and service provision may be better served should both get housed in the same super-organization. Code.org could offer another Hour of Code in another year, and next time further align marketing, tutorial access, and school delivery.

Time Cures All Woes

The incentive for the Hour of Code was that too many people lack a basic understanding of computing. However, in the existing tutorials, such as the Angry Birds tutorial or the MIT App Development tutorial, the programming is different from what was taught in introductory computer science thirty years ago. Instead, the student has stylized, visual instructions and not a general-purpose programming language. This may help to explain why students who completed an Hour of Code tutorial did not show improved skills for coding when they were later asked to answer the traditional programming questions.

Increasingly people are able to get computational devices to perform useful tasks by engaging a highly stylized programming interface. In the 1980s, human-computer interaction experts showed that spreadsheets were special declarative programming systems for accountant-type activities and that their purpose was extremely powerful (Lewis, 1985). The argument for new computer science courses for students who major in the arts is that those courses should focus on helping students develop media objects with computers whether or not the 'programming tool' fits the classical model of a programming language (Forte & Guzdial, 2005). From these and many other examples, one sees that the trend is to have people achieve every day results with computers in a way that is a restricted kind of programming. That is exactly what Hour of Code tutorials try to teach students.

When a person uses his online calendar to schedule a meeting every Monday at 11 a.m. for the next two months, the person has implemented a simple computer program in the highly stylized language of online calendars. When a person tells his GPS system that he wants to go from home to the store and then to work, to avoid highways and traffic, but otherwise take the shortest route, the person has given a computer instructions in a stylized language for GPS. As such applications spread, more people become engaged in some basic and invisible 'programming'. The intention of Hour of Code tutorials is to make students aware that learning how to code is very important and to keep them open-minded to learning coding. Those who understand loops and conditionals are ahead of others in their ability to master the world everyone faces. However, whether or not an Hour of Code is taught to everyone, this ability to 'program devices' will become increasingly pervasive. An Hour of Code might help people understand and control their world of devices.

Conclusion

The Code.org video of February 2013 reached millions of viewers, and the Hour of Code is claimed by its organizers to be a success. The authors conducted a study by asking a group of undergraduate students to participate in an Hour of Code tutorial and then surveying them about their attitudes toward programming and their understanding of programming knowledge. The results indicate the positive impact of the Hour of Code on students' attitudes toward programming. However, completing an Hour of Code tutorial alone does not necessarily impact students' skills for coding, which suggests that a combination of online tutorials with a traditional computer science lecture may be necessary to improve students' coding knowledge.

Motivating students is a challenge for any future Hour of Code tutorials. This depends on whether students are working toward a credential from a school or seeking external rewards, such as connection with an employer or financial gain. Code.org wants to see a computer science curriculum that is mandatory in schools and run by appropriately trained teachers. The Hour of Code was basically a marketing effort. Further connecting this marketing effort to the goals of mandating computer science education and training computer science teachers might help volunteers better understand how they can support future Hour of Code initiatives.

References

Akopian, D., Melkonyan, A., Golgani, S., Yuen, T., & Saygin, C. (2013). A template-based short course concept on android application development. *Journal of Information Technology Education: Innovations in Practice, 12*(1), 13-28. Retrieved from http://www.jite.org/documents/Vol12/JITEv12IIPp013-028Akopian1196.pdf

Al-Wakeel, S. S. (2001). *Innovation in computer education curriculum for the computerization of Saudi Arabia: a model for developing countries.* Paper presented at the Frontiers in Education Conference, 2001. 31st Annual.

Al-Wakeel, S. S. (2013). *Creativity and innovation in computer education: A solid foundation for the Saudi society's knowledge.* Paper presented at the First Workshop on Computer and Information Sciences: Computer Education Quality and Innovation, Tabuk, Saudi Arabia, , Nov 21-22.

Ali, A., & Smith, D. (2014). Teaching an introductory programming language in a general education course. *Journal of Information Technology Education: Innovations in Practice, 13*, 57-67. Retrieved from http://www.jite.org/documents/Vol13/JITEv13IIPp057-067Ali0496.pdf

Apiola, M., & Tedre, M. (2012). New perspectives on the pedagogy of programming in a developing country context. *Computer Science Education, 22*(3), 285-313.

Apone, K., Bers, M., Brennan, K., Franklin, D., Israel, M., & Yongpradit, P. (2015). *Bringing grades K-5 to the mainstream of computer science education.* Paper presented at the Proceedings of the 46th ACM Technical Symposium on Computer Science Education.

Bouhnik, D., & Giat, Y. (2009). Teaching high school students applied logical reasoning. *Journal of Information Technology Education: Innovations in Practice, 8*(1), 1-16. Retrieved from http://www.jite.org/documents/Vol8/JITEv8IIP001-016Bouhnik681.pdf

Chen, C., & Rada, R. (1996). Interacting with hypertext: A meta-analysis of experimental studies. *Human-Computer Interaction, 11*(2), 125-156.

Cheng, T. K., Jayasuriya, M., & Lim, J. (2010). Removing the fear factor in Programming. *The Python Papers Monograph, 2*, 1-9.

Daly, T. (2009). Using introductory programming tools to teach programming concepts: A literature review. *The Journal for Computing Teachers, Fall*. Retrieved from http://www.iste.org/jct

Dann, W. P., Cooper, S., & Pausch, R. (2011). *Learning to program with Alice (w/CD ROM)*: Prentice Hall Press.

Elliott Tew, A., Dorn, B., & Schneider, O. (2012). Toward a validated computing attitudes survey. *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, 135-142.

Ford, M., & Venema, S. (2010). Assessing the success of an introductory programming course. *Journal of Information Technology Education: Research, 9*(1), 133-145. Retrieved from http://www.jite.org/documents/Vol9/JITEv9p133-145Ford810.pdf

Forte, A., & Guzdial, M. (2005). Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses. *IEEE Transactions on Education, 48*(2), 248-253.

Franke, B., & Osborne, B. (2015). *Decoding CS principles: A curriculum from Code. org.* Paper presented at the 46th ACM Technical Symposium on Computer Science Education.

Goel, S., & Kathuria, V. (2010). A novel approach for collaborative pair programming. *Journal of Information Technology Education: Research, 9*(1), 183-196. Retrieved from http://www.jite.org/documents/Vol9/JITEv9p183-196Goel314.pdf

GuideStar. (2014). *GuideStar exchange report on Code.org*. Retrieved from http://www.guidestar.org/organizations/46-0858543/code-org.aspx

Guynn, J. (2013, Feb. 26). Silicon Valley launches campaign to get kids to code. *Los Angeles Times*.

Hanks, B., Fitzgeral, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computers Science Education, 21*(2), 135-173.

Hu, M., Winikoff, M., & Cranefield, S. (2012). Teaching novice programming using goals and plans in a visual notation. *Proceedings of the Fourteenth Australasian Computing Education Conference-Volume 123*, pp. 43-52.

Kay, R. H. (1994). An exploration of theoretical and practical foundations for assessing attitudes toward computers: The computer attitude measure (CAM). *Computers in Human Behavior, 9*(4), 371-386.

Koohang, A. A. (1989). A study of attitudes toward computers: Anxiety, confidence, liking, and perception of usefulness. *Journal of Research on Computing in Education, 22*(2), 137-150.

Lee, M., Pradhan, S., & Dalgarno, B. (2008). The effectiveness of screencasts and cognitive tools as scaffolding for novice object-oriented programmers. *Journal of Information Technology Education: Research, 7*(1), 61-80. Retrieved from http://www.jite.org/documents/Vol7/JITEv7p061-080Lee332.pdf

Lewis, C. (1985). *Extending the spreadsheet interface to handle approximate quantities and relationships.* Paper presented at the CHI'85 Human Factors in Computing Systems, San Francisco, California.

Long, J. (2007). Just for fun: Using programming games in software programming training and education. *Journal of Information Technology Education: Research, 6*(1), 279-290. Retrieved from http://www.jite.org/documents/Vol6/JITEv6p279-290Long169.pdf

Malan, D. J., & Leitner, H. H. (2007). *Scratch for budding computer scientists. ACM SIGCSE Bulletin, 39*(1), 223-227.

Malliarakis, C., Satratzemi, M., & Xinogalos, S. (2013, Sept 19-21). *Towards a new massive multiplayer online role playing game for introductory programming.* Paper presented at the BCI'13, Thessaloniki, Greece.

Maloney, J. H., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE), 10*(4), 16.

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with scratch. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, pp. 367-371.

Moor, B., & Deek, F. (2006). On the design and development of a UML-based visual environment for novice programmers. *Journal of Information Technology Education: Research, 5*(1), 53-76. Retrieved from http://www.jite.org/documents/Vol5/v5p053-076Moor29.pdf

Nikula, U., Sajaniemi, J., Tedre, M., & Wray, S. (2007). Python and roles of variables in introductory programming: Experiences from three educational institutions. *Journal of Information Technology Education: Research, 6*(1), 199-214. Retrieved from http://www.jite.org/documents/Vol6/JITEv6p199-214Nikula269.pdf

Pendergast, M. (2006). Teaching introductory programming to IS students: Java problems and pitfalls. *Journal of Information Technology Education: Research, 5*(1), 491-515. Retrieved from http://www.jite.org/documents/Vol5/v5p491-515Pendergast128.pdf

Powell, L. M. (2015). Evaluating the effectiveness of self-created student screencasts as a tool to increase student learning outcomes in a hands-on computer programming course. *Information Systems Education Journal, 13*(5), 106.

Rada, R. (2013, Nov 20-21). *Opportunities in CS education.* Paper presented at the First Workshop on Computers and Information Sciences: Computer Education Quality and Innovation, Tabuk, Saudi Arabia.

Rajala, T., Laakso, M.-J., Kaila, E., & Salakoski, T. (2008). Effectiveness of program visualization: A case study with the ViLLE tool. *Journal of Information Technology Education: Innovations in Practice, 7*. Retrieved from http://www.jite.org/documents/Vol7/JITEv7IIP015-032Rajala394.pdf

Raub, A. C. (1981). *Correlates of computer anxiety in college students*. (Doctoral dissertation: University of Pennsylvania) ProQuest paper AAI8208027.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., . . . Silverman, B. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60-67.

Rizvi, M., Humphries, T., Major, D., Jones, M., & Lauzun, H. (2011). A CSO course using scratch. *Journal of Computing Sciences in Colleges, 26*(3), 19-27.

Roy, K. (2012). App inventor for android: Report from a summer camp. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, pp. 283-288.

Rubio, M. A., Romero-Zaliz, R., Mañoso, C., & Angel, P. (2015). Closing the gender gap in an introductory programming course. *Computers & Education, 82*, 409-420.

Sanders, M. E. (2008). STEM, STEM Education, STEMmania. *The Technology Teacher, 68*(4), 20-26

Sykes, E. R. (2007). Determining the effectiveness of the 3D Alice programming environment at the computer science I level. *Journal of Educational Computing Research, 36*(2), 223-244.

Venkatesh, V. (1999). Creation of favorable user perceptions: Exploring the role of intrinsic motivation. *MIS Quarterly*, 239-260.

Wilson, C. (2015). Hour of code---A record year for computer science. *ACM Inroads, 6*(1), 22-22.

Wolber, D. (2011). App inventor and real-world motivation. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 601-606).

Wolber, D., Abelson, H., Spertus, E., & Looney, L. (2011). *App Inventor.* Sabastopol, CA, USA: O'Reilly Media, Inc.

Appendix

We would like to examine students' responses in order to detect changes, if any, in their attitudes toward programing and coding skills. Therefore, we use the same questions in both pre and post survey and compare their responses. Two additional questions only show up in the post survey to ask their feedback of taking the tutorial.

The sample of survey questions:

1) First, please enter a PIN number (Please note: You need to enter the SAME number when you take the pre and post-survey so that your responses in the pre and post survey will be matched. You choose your own PIN)

2) Have you ever taken any programming courses?
   - Yes
   - No

3) What's your experience with programming?
   - Less than 1 year
   - 2 to 3 years
   - 4-5 years
   - 5+ years

4) To what extent do you agree or disagree with the following statement: Everybody in this country should learn how to program a computer because it teaches you how to think.

   - Strongly Disagree
   - Disagree
   - Neither agree nor disagree
   - Agree
   - Strongly Agree

5) How likely are you to take a programming course?
   - Very likely
   - Moderately likely
   - Slightly likely
   - Not at all likely

6) Which of the lettered choices is equivalent to the following decision?

```
if x> 10 then
        if  x>y then
                Print "x"
        endif
endif
```

   a. If x>10 or y>10 then print "x"
   b. If x>10 and x>y then print "x"
   c. If y>x then print "x"
   d. If x>10 and y>10 then print "x"

7) In the following pseudocode, what is printed?

```
g = 6
h = 4
while g < h
      g = g+1
endwhile
print g, h
```

a. nothing
b. 4 6
c. 5 6
d. 6 4

8) * Did you enjoy the tutorial provided by code.org?
- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly agree

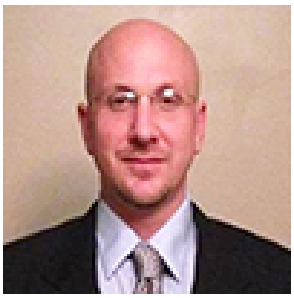
9) * Completing the tutorial positively changed your attitude towards programming?
- Strongly disagree
- Disagree
- Neutral
- Agree
- Strongly agree

10) In the space below, please share any additional comments regarding programming.

* The questions will only appear in the post-survey

Biographies

Jie Du is an Assistant Professor in the School of Computing and Information Systems at Grand Valley State University. She received her BS in Information Systems from Southwest Jiaotong University, MS and PhD in Information Systems from the University of Maryland, Baltimore County. Her main research interests include artificial intelligence and decision support systems.



Hayden Wimmer is an Assistant Professor in the Department of Information Technology at Georgia Southern University. He has a Ph.D. from the University of Maryland Baltimore County in Information Systems based in data mining and artificial intelligence applied to financial data. He also holds an M.S. is in Information Systems from UMBC, an M.B.A. from the Pennsylvania State University, and a B.S.in Information Systems from York College of PA. Dr. Wimmer has multiple journal publications related to multi-agent systems, artifi-

cial intelligence, data science, and I.S. education; and serves in various editorial capacities including co-editor in chief, board member, and reviewer of various journals and conferences and is a member of the Association of Information Systems.

Roy Rada has a B.A. in Psychology from Yale University, a M.D. from Baylor College of Medicine, and a Ph.D. in Computer Science from Univ. Illinois. He was a Professor of Computer Science at the University of Liverpool where his group created the Many Using and Creating Hypermedia system, and many journal articles were published about the applications of that system to education and training. After that he was the Virtual University Academic Officer at Washington State University and led that universities efforts to create a virtual university. At the same he published a book called 'Virtual Education Manifesto'. He later moved to the University of Maryland Baltimore County as the founding Director of its Online Master's Degree in Information Systems. He and his students have extensively studied many aspects of the use of information technology in education.