CHILDREN'S PERCEPTIONS OF WHAT COUNTS AS A PROGRAMMING LANGUAGE *

Colleen Lewis
Harvey Mudd College
Claremont, CA 91711
lewis@cs.hmc.edu

Sarah Esper
University of California, San Diego
La Jolla, CA 92093
sesper@eng.ucsd.edu

Victor Bhattacharyya
Harvey Mudd College
Claremont, CA 91711
vbhattachary@hmc.edu

Noelle Fa-Kaji
Scripps College
Claremont, CA 91711
noelle.fa-kaji@scrippscollege.edu

Neftali Dominguez
Harvey Mudd College
Claremont, CA 91711
ndominguez@hmc.edu

Arielle Schlesinger
Pitzer College
Claremont, CA 91711
contact.aribea@gmail.com

ABSTRACT

An educational programming language may be more accessible, less frustrating, and more rewarding for young students. However, if a student thinks that the educational language does not constitute actual computer programming, it is hard to build interest in the subject and confidence in their programming skills. We conducted a study in a summer enrichment program for academically high-achieving students entering the sixth grade. Students were interviewed about their perception of various computer programming environments and our analysis of their reasoning helps us to better understand students' tacit assumptions about computer programming. Half of the students were selected to complete a worksheet showing the connections between the programming language Scratch, Java, C++, and Python. Using linear

regression we found that there were no statistically significant differences between students who did and did not complete this worksheet in terms of confidence and perception.

1. INTRODUCTION

1.1 Background

There has been international interest in exposing more students to computer science before college. One possible goal of such efforts is to build students' interest in pursuing computer science at the college level. These efforts sometimes teach students to use programming languages that are used in the software industry (e.g., Java, Python, C++) and other times teach students programming languages designed for educational purpose (e.g., Scratch [12], Alice [2], Code Spells [4,5], Lego Mindstorms [7]). Many of these programming languages designed for educational purposes (or "educational languages") look quite different from text-based industry programming languages (or "professional languages"). The programming language LOGO [6, 14] was designed for education, but because it has a command prompt and textual commands, it looks more like many professional languages (see Figure 1). Previous research found that students who learned the LOGO programming language developed higher confidence in their programming ability than students using the Scratch programming language [9].

1.2 Motivation

It is important to understand the benefits and tradeoffs of using an educational language. An educational language may be more accessible, less frustrating, and more rewarding for young students, but it may not accomplish the goal of building their interest and confidence in computer programming if students perceive the language as distinct from computer programming. Additionally, if a student is performing well but does not believe they are learning computer science they may doubt their authority and the legitimacy of their knowledge.

1.3 Research Questions

- Question 1: Does completing a worksheet that shows how the programming language Scratch relates to Java, C++, and Python:
  - increase students' confidence and interest in computer programming?
  - convince students that Scratch is a programming language and will help them when learning other programming languages?
    and do these patterns differ between boys and girls?
- Question 2: What do students think counts as a programming language?

1.4 Overview of Study and Findings

We conducted a study in a summer enrichment program for academically high achieving students entering the sixth grade. Half of the students were selected to complete

a worksheet showing the connections between the programming language Scratch, Java, C++, and Python. Using linear regression we found that there were no statistically significant differences between students who did and did not complete this worksheet in terms of confidence or perception. Students were also interviewed about their perception of various computer programming environments and an analysis of their reasoning helps us to better understand students' tacit assumptions about computer programming.

2. PREVIOUS RESEARCH

2.1 Weaknesses in Building Confidence Using Scratch

Previous research [9] compared sixth-grade students' confidence and competence after completing a computer programming enrichment course in either Scratch [12] or LOGO [6, 14]. This research found that students who used the programming language Scratch performed better on assessments of their programming competence, but were less confident in their programming ability. Using a Mann-Whitney-Wilcoxon test, this research found that students who used the programming language LOGO responded more positively to the prompt "I am good at writing computer programs" ($z=-2.02$; $p=0.044$) [9].

From this previous research, we hypothesize that a student's expectations of what constitutes computer programming shapes their perception of their ability to write computer programs. Consequently, it is possible that students learning Scratch did not realize they were programming. However, students received consistent exposure to language linking Scratch to computer programming. We perceive that a student's expectations of computer programming is more complicated and deeply ingrained.

Simply telling students that Scratch is programming may not have been a large enough intervention to alter expectations of computer programming. In the current study we seek to build upon this previous work to see if we can better understand students' expectations of programming. As such, we developed an interview task to uncover what attributes explicitly contribute to students' expectations of computer programming (described in Section 4.2).

2.2 Importance of Building Students' Confidence

Building confidence and interest might be particularly important for young women, who are underrepresented within computer science [3]. A study of over 24 thousand students found that even though the differences between the mathematical performance of boys and girls were negligible, girls were reported to be less confident and less interested in mathematics [1]. Research has shown that students' perception of their computer science ability is consequential to their future career choice, and of those who leave the computer science major in college, women had higher grades than their male colleagues before they quit [17].

3. RESEARCH CONTEXT

3.1 Overview of the Class

The course "Creating Music, Movies, and Games with Computers" was offered as part of a summer enrichment program sponsored by a university, and has been the focus of multiple studies [9, 10, 11, 16]. The course lasted three weeks in the summer of 2013. There were two sections of the course: one in the morning and one in the afternoon. The course met four days per week, totaling to 36 hours of instruction per section. Each day the students met for three hours, consisting of two 90-minute periods of instruction separated by a 30-minute break. The students used a set curriculum to program in Scratch for the majority of the course (28.5 hours) and completed a final project of their own design in Scratch (7.5 hours). To review the concepts from the day before, students began each class with an individual 10-15 minute paper-based warm-up. For the remainder of the time, students followed an online curriculum [8], which included instructional videos to introduce new concepts, related programming assignments, and online quizzes to assess students' understanding.

3.2 Description of the Teaching Staff

The course had six teachers, a college computer science professor, a computer science education PhD candidate, and four undergraduate students with at least two courses of computer science experience. Five of the six teachers were women. Each day teachers attended to a variety of tasks: helping students, one-on-one interviews with students, grading homework and quizzes, and classroom preparation for the following day. Many of these tasks removed the teachers from active engagement with students working through the curriculum. On average there were two or three teachers available for active engagement with students during class time.

3.3 Description of the Students

The summer enrichment program at the focus of our study is intended for academically advanced students. Of students who provided standardized test scores in a previous offering of this course, 100% of students received a score of "Advanced" for Mathematics and 95% were "Advanced" in English-Language Arts respectively [11]. This is in contrast to the 42% and 36% of student in the state who were classified as "Advanced" on the same exams [11]. In total, 37 students participated in the research: 19 in the morning section and 18 in the afternoon section. The morning section had 8 boys and 11 girls while the afternoon section was evenly split, with 9 boys and 9 girls. The provided course description stated that the program was intended for students with no previous programming experience. Interviews on the first day revealed that 16 of the 38 students said they had had previous experience. However, at least 5 of these were for less than an hour, and many of the activities students described as programming included typing or using other software.

4. METHODS

4.1 Provided Definition of Programming and Programming Language

On the first day, before any assessments or interviews, students were introduced to the idea of computer programming. The lead instructor described that computer programming is "writing rules or instructions for the computer to follow" and that "to boss a computer around we need to learn a language that the computer understands." Scratch was introduced as a programming language, which was defined as "a language the computer understands." In the back of the classroom, a display showed nine programming languages. Students' attention was drawn to this classroom display and the goal was to help students see the connection between Scratch and other programming languages. This introduction on the first day may have made it harder to identify students' initial expectations of computer science; when asked "what is computer programming?" students often repeated the words of the lead instructor.

4.2 Programming Language Card Sort

On the first day of the course, three instructors conducted one-on-one, audio-recorded interviews with students regarding their perceptions of different programming languages. Students were asked to sort cards into one of three categories: definitely programming, somewhat like programming, and not programming. Each card was a picture pasted on 5x8 index card. Most of the pictures showed code in various languages and programming environments (see Figures 1 and 2), but we also included some other cards like a picture of a terminal window with a meaningless set of characters (see Figure 1), and a screenshot of green symbols from The Matrix (not shown).

We performed both quantitative and qualitative analyses of data collected during these interviews. First, we calculated summative statistics for how students sorted each of the cards. Second, we mined students' justification for their sorting to better understand their tacit assumptions about computer programming.

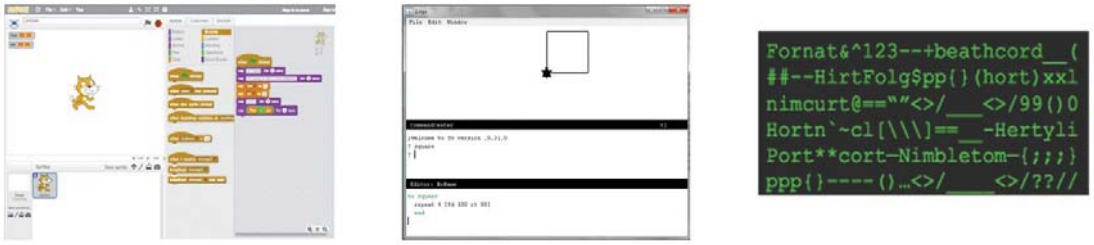

Figure 1. Images from the card sort: Scratch (left), LOGO (center), and a meaningless set of symbols (right).

4.3 Students' Description of Scratch

For another measure of students' impression of the Scratch programming language, we asked students on the first homework assignment to answer the following question: "How would you describe Scratch to a friend that had never seen it?" We mined the text of their answers to better understand students' beliefs about Scratch after the first day of

class. This provided an initial judgment that we could use to see how their perception changed throughout the course.

4.4 Translation Worksheet

4.4.1 Intervention Overview

We gave the students a seven-page homework assignment wherein students were scaffolded to translate Scratch into three professional programming languages (C++, Java, and Python). There were also survey questions at the beginning and end of the homework.

4.4.2 Assignment to Treatment and Control

This homework was given to approximately half of the students on day 6 of the course, the treatment group (19 students in the treatment group out of 37). The remaining students, the control group, received a review homework on day 6. After the study was completed, the treatment group completed the review homework and the control group completed the language translation homework.

Recall that each class began with an individual 10-15 minute, paper-based assessment that we referred to as a warm-up. Students were split into treatment and control groups based on their average warm-up scores from the first five days of the course; this includes 4 warm-ups in total as there was no warm-up on the first day of the course. We sorted the students by this cumulative score within each section. Then for adjacent each pair, we assigned one student to the treatment and one student to the control. The treatment group had 7 boys and 11 girls while the control group had 10 boys and 8 girls.

4.4.3 Translation Worksheet Content

The styles of questions in this assignment varied. Some questions presented functionally-equivalent code in the four languages inside of a four-cell table; under the code there were boxes for the student to add a line of code in each language. The task involved mimicking the syntax demonstrated in the provided lines of code. For example, one question, shown in Table 1, asked the student to add a fifth print statement in each language that prints their name. The intention was to show how the languages map onto one another. Additional questions asked students to point out syntax differences between the languages, such as semicolon placement. In addition to these questions, there were short response questions and multiple choice questions.

Table 1. Example problem from the translation worksheet.

Scratch	C++	Java	Python

	`cout << "Hello";`
	`cout << "my";`
	`cout << "name";`
	`cout << "is";`
		`System.out.print("Hi");`
		`System.out.print("my");`
		`System.out.print("name");`
		`System.out.print("is");`
			`print "Hello"`
			`print "my"`
			`print "name"`
			`print "is"`

Two questions on the homework asked the students to log when they started and finished the worksheet. We used the data from these questions to determine that the language translation worksheet on average took about 32 minutes, with the control group taking an average of 31 minutes and the treatment group 33 minutes. When we broke down the group into those who took 30 minutes or less versus those who took more than 30 minutes, we found both groups did equally well on the assignment (average score of 88 percent for slower students versus 90 percent for quicker students) leading us to believe time spent on the worksheet had little to do with performance ($p>0.05$).

4.5 Surveys on Eighth Day

During the eighth day of class, we asked the students to fill out a survey. On the survey, students were asked to rate the following statements using a 5-level Likert scale from "strongly agree" to "strongly disagree."

- "I think I am good at programming"
- "By learning Scratch, I have been learning computer programming"
- "It will be easier for me to learn new programming languages because I know Scratch"
- "I would like to be a computer scientist when I grow up"
- "Scratch is a computer programming language"

5. RESULTS

5.1 Research Question #1: Effects of Translation Worksheet

On the eighth day only students in the treatment group had completed the translation worksheet. We used students' responses to the survey described in Section 4.5 to evaluate components of research question 1. We coded students' responses from the five-level Likert scale as: "strongly agree" = 5, "agree somewhat" = 4, "neutral" = 3, "disagree somewhat" = 2, "strongly disagree" = 1. For each Likert prompt shown in section 4.5 we computed a linear regression. Although we anticipated greater endorsement for each prompt from students in the treatment group, this was only the case for two of the five prompts and none of the differences were statistically significant at the five-percent level. Students in both the control and treatment were likely to select "agree somewhat" or

"strongly agree" for each prompt. It is likely that ceiling effects reduced our ability to detect any differences between the control and treatment.

There were interesting differences in how boys and girls responded to the Likert questions, independent of whether they were in the treatment or control. Although there were not statistically significant differences in boys and girls performance on daily warm-ups, on average boys responded more positively to the statement "I think I am good at programming." The average response for girls was 3.63 and the average response for boys was 4.65 ($p<0.001$). This result is consistent with other studies in which women are less confident in their ability [1], despite the fact that the course was designed to provide all students access to identities as computer scientists [16]. More boys responded positively to the statement "I would like to be a computer scientist when I grow up." The average response for girls was 2.97 and the average response for boys was 3.47. This is consistent with the underrepresentation of women in computer science, however this difference was not statistically significant ($p>0.05$).

Although there were no statistically significant differences between the treatment group and control group for the question "I would like to be a computer scientist when I grow up," there was an interaction between gender and treatment group. Girls in the treatment group responded more positively to the question than girls in the control group (3.50 vs. 2.25; $p<0.01$). However, boys in the treatment group responded more negatively to the question than boys in the control group (2.86 vs. 3.90; $p=0.015$).

5.2 Research Question #2: Perception of What Counts as Programming

5.2.1 How Students Described Scratch in the First Homework Assignment

Recall that the first night there was a question on the homework that asked students how they would describe Scratch to a friend. Possibly because of the visual nature of Scratch, many of the students described Scratch as "a really cool website where you … [are] learning to animate." One student described it as "a computer program that helps kids learn how to program computers in a fun and easy way." Another said, Scratch is "a simple programming language that is drag and drop." Only two students described Scratch as a programming language verbatim, though most of the students' responses could be generalized as recognizing that they were learning to control and create things using a computer. Another student described Scratch as a "program where you can make music, movies, and games," which appears to be taken directly from the course title: "Making Music, Movies, and Games with Computers." Taken together, these results do not suggest that "programming language" was central to students' classification of Scratch.

5.2.2 Students' Card Sorting

Recall that students sorted cards into the following categories: definitely programming, somewhat programming, and not programming. The goal of the card sort was to better understand students' perception of what counts as a programming language.

Students universally reported that Scratch was a programming language. It is possible that students said Scratch was a programming language because they were aware

of our desires for them to believe that [15]. This makes it impossible to distinguish whether the students truly believe Scratch is a programming language or if they were simply providing us with the answers we wanted to hear. Similarly, all but one student believed that App Inventor [18] (as shown in Figure 2, right) was programming and many students justified their decision by explaining its similarity to Scratch.

A common theme was that students believed a particular programming environment was not a programming environment and was instead a game. One student described Code Spells [4, 5] (as shown in Figure 2, left): *"Well basically, that looks like a game you could play online, and when you're playing a game, you're not programming, you're just playing a game."* Similarly, thirteen students believed that Lego Mindstorms [7] (as shown in Figure 2, center) was not programming and many reported that it looked like a "game" or a "video game."

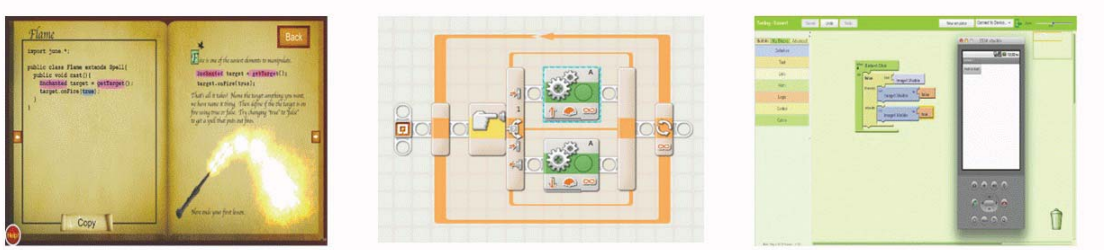

Figure 2: Images from the card sort: Code Spells (left), Lego Mindstorms (center), and App Inventor (right).

Students' justifications for what was definitely programming suggest that they have expectations of programming that may be distinct from characteristics of Scratch. For example, a student described Eclipse as "looking more official" and another student said that Eclipse "looks like something that would be programming." Another student described App Inventor as "too simple." Eleven students said that the picture of green symbols from the Matrix was definitely programming. Another sixteen students thought it was somewhat like programming and only nine students believed it was not programming. Justifications included that it was "complicated" and that "there's just a bunch of lights, and that's what the inside of a computer looks like."

6. CONCLUSION

In our research we sought to address the following two questions:

- Question 1: Does completing a worksheet that shows how the programming language Scratch relates to Java, C++, and Python:
  - increase students' confidence and interest in computer programming?
  - convince students that Scratch is a programming language and will help them when learning other programming languages?

  and do these patterns differ between boys and girls?
- Question 2: What do students think counts as a programming language?

We were not able to reject the null hypothesis in investigating Research Question 1. A possible reason why the intervention did not produce a measurable difference in perceptions of Scratch as a programming language in our context is that the instructors were already focused on helping students understand that they were engaging in practices of computer science using a computer programming language [16]. However, boys reported that they were more confident in the programming ability despite the fact that differences between boys' and girls' performance was not statistically significant. This is also surprising given that everyday students interacted with the five women teachers. Additionally, throughout the course we displayed photographs of, and introduced, a diverse group of accomplished computer scientists, which included many women [16].

In investigating Research Question 2, our analysis provided insights into students' tacit expectations of computer programming as "complex" and distinct from games. These expectations may be relevant to increasing students' confidence in their programming ability.

7. ACKNOWLEDGEMENTS

We would like to thank the students and staff of the summer enrichment program. The research reported here was supported in part by a grant from the National Science Foundation (Award #1044106). The opinions expressed are those of the authors and do not represent the views of the National Science Foundation.

8. REFERENCES

[1] Catsambis, S. (1994). A Path to Math: Gender and Racial-ethnic Differences in Mathematics Participation from Middle School to High School. Sociology of Education. *67 (3)*, 199-215.

[2] Cooper, S. (2010) The Design of Alice. ACM Transactions on Computing Education. **10(4)**, 1-16.

[3] Ensmenger, N. (2010). Making Programming Masculine. In Gender Codes: Why Women Are Leaving Computing, edited by Thomas Misa, Wiley.

[4] Esper. S., Foster, S. R., & Griswold, W. G. (2013). On the nature of fires and how to spark them when you're not there. ACM SIGCSE Bulletin. *44(1)*, 305-310.

[5] Esper, S., Foster, S. R., Griswold, W. G. (2013). CodeSpells: embodying the metaphor of wizardry for programming. In ITiCSE '13. ACM, New York, NY, USA, 249-254.

[6] Haas, G. (2013). TG (Version 9.36) [Software]. Available from http://www.bfoit.org/itp/install.html

[7] Klassner, F. (2002). A case study of LEGO Mindstorms' suitability for artificial intelligence and robotics courses at the college level. ACM SIGCSE Bulletin. *33(1)*. 8-12.

[8] Lewis, C. M., et al. (2013). Online curriculum. Retrieved from http://colleenmlewis.com/scratch

[9] Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch, ACM SIGCSE Bulletin. *41(1)*, 346-350.

[10] Lewis, C. M. (2011). Is pair programming more effective than other forms of collaboration for young students? Computer Science Education. *21(2)*, 105-134.

[11] Lewis, C. M., & Shah, N. (2012). Building Upon and Enriching Grade Four Mathematics Standards with Programming Curriculum. ACM SIGCSE Bulletin. *43(1)*, 57-62.

[12] Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2010). Learning computer science concepts with Scratch. International Computer Science Education Research Workshop. Aarhus, Denmark. 69-76.

[13] Nunkoosing, K. (2005) The problems with interviews. Qualitative Health Research *15 (5)*, 698-706.

[14] Papert, S. (1980). Mindstorms: Children, Computers, and Powerful Ideas. New York: Basic Books, Inc.

[15] Partington, G. (2001). Qualitative research interviews: Identifying problems in technique. *Issues In Educational Research*, 11(2), 32-44. http://www.iier.org.au/iier11/partington.html

[16] Shah, N., Lewis, C. M., Caires, R., Khan, N., Qureshi, A., Ehsanipour, D., & Gupta, N. (2013). Building Equitable Computer Science Classrooms: Elements of a Teaching Approach. ACM SIGCSE Bulletin. *44(1)*, 263-268.

[17] Strenta, A. C., Elliott, R., Adair, R., Matier, M., & Scott, J. (1994). Choosing and Leaving Science in Highly Selective Institutions. Research in Higher Education, 35(5), 513-547.

[18] Wobler, D. (2011) App Inventor and real-world motivation. ACM SIGCSE Bulletin. *42(1)*, 601-606.