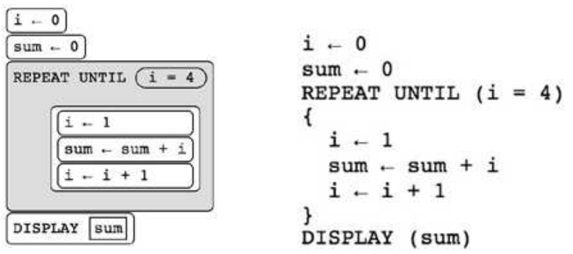
Blocks or Text? How Programming Language Modality Makes a Difference in Assessing Underrepresented Populations

David Weintrop, University of Maryland, weintrop@umd.edu
Heather Killen, University of Maryland, hkillen@umd.edu
Baker Franke, Code.org, baker@code.org

Abstract: Broadening participation in computing is a major goal in contemporary computer science education. The emergence of visual, block-based programming environments such as Scratch and Alice have created a new pathway into computing, bringing creativity and playfulness into introductory computing contexts. Building on these successes, national curricular efforts in the United States are starting to incorporate block-based programming into instructional materials alongside, or in place of, conventional text-based programming. To understand if this decision is helping learners from historically underrepresented populations succeed in computing classes, this paper presents an analysis of over 5,000 students answering questions presented in both block-based and text-based modalities. A comparative analysis shows that while all students perform better when questions are presented in the block-based form, female students and students from historically underrepresented minorities saw the largest improvements. This finding suggests the choice of representation can positively affect groups historically marginalized in computing.

Introduction

In an effort to broaden participation in computing and give learners a more accurate sense of the field of computer science, and how it involves more than just programming, a new Advanced Placement (AP) course was created in the United States. The new course, titled *AP Computer Science Principles* (AP CSP), was taught nationally for the first time in the 2016-2017 school year and covers seven big ideas in computing: Creativity, Abstraction, Data, Algorithms, Programming, Internet, and Global Impacts. To help foster an inclusive learning experience, the course is programming language agnostic, allowing teachers to choose the technologies and programming environments for instruction. This means some learners go through the year-long high school course using block-based programming environments like Scratch, while students in other classrooms might use conventional text-based programming languages like Python. This presents a challenge for the organization tasked with creating a single written summative assessment to be administered nationally at the conclusion of the course: how do you create a written assessment for a computer science class when you do not know the programming language, or even the programming modality (text-based or block-based) that learners used? Further, because learners can use either modality, what can this assessment tell us about the role of programming modality towards the goal of broadening participation in computing? Given that a goal of the course is to broaden participation in the field, the assessment must reflect both the programming plurality welcomed in the design of the curriculum as well as the equity-oriented priorities of the course.



```
i ← 0
sum ← 0
REPEAT UNTIL (i = 4)
  i ← 1
  sum ← sum + i
  i ← i + 1
DISPLAY sum
```

```
i ← 0
sum ← 0
REPEAT UNTIL (i = 4)
{
  i ← 1
  sum ← sum + i
  i ← i + 1
}
DISPLAY (sum)
```

Figure 1. The block-based (left) and text-based (right) modalities from the AP Computer Science Principles exam.

In response to this challenge, the Development Committee for the AP CSP course invented a custom pseudocode and included both text-based and block-based forms of questions on their exam (Figure 1). In this way, students are not rewarded or penalized for using one type of programming tool or another during the school year. However, there are concerns associated with this approach. By creating a new pseudo-language that no student has used during the course, it is unclear how students will perform. Further, many of the identified benefits associated with block-based tools are absent from the block-based form of the assessment. For example, the block-based pseudocode does not use colors, present commands using natural language expressions, or differentiate block-shape by use. Further, the assessment only asks questions related to program comprehension, thus students are not asked to compose programs which is another a perceived affordance of block-based tools (Weintrop & Wilensky, 2015a).

The decision to *both* invent a pseudocode language that has two modalities *and* to present questions on the exam in both modalities is consequential for any student but especially given the goal of welcoming students from historically underrepresented populations to the field of computer science. If the questions asked on the summative exam do not align with the features of the learning environments that have been found to support students from historically underrepresented populations, then the course has the potential to perpetuate existing inequalities, rather than solve them. Understanding the outcomes from the assessment approach used for the written AP CSP exam, particularly with respect to the goal of broadening participation in the field of computer science, is the focus of this paper. Stated more explicitly, this paper answers the following research questions:

RQ 1: How do students perform on questions asked in an unimplemented text-based pseudocode compared to an isomorphic block-based pseudocode on a written computer science assessment?

RQ 2: How does modality (block-based versus text-based) affect students from historically underrepresented populations on a standardized computer science assessment?

To answer these questions, we designed a 20-question assessment that asks questions using both the block-based and text-based version of the pseudocode and embedded it as part of an AP CSP curriculum. Over 5,000 students took the assessment at schools across the United States. Using the responses, we are able to shed light on the stated research questions and advance our understanding of how the design of programming languages and assessments can impact learner outcomes and the goal of broadening participation in computing.

Literature review

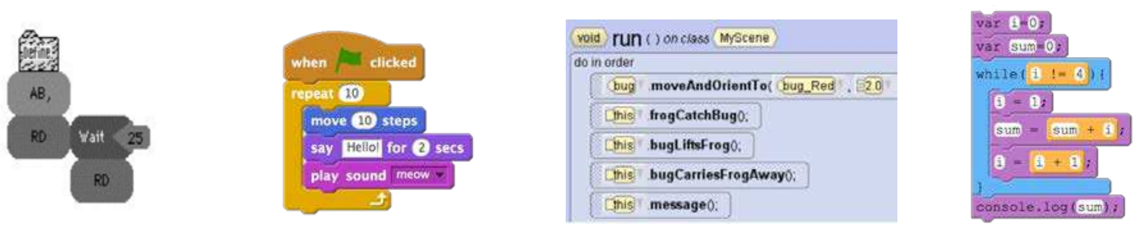
Broadening participation in computing

Despite the growing presence of computing in society, the field of computer science still struggles with issues of underrepresentation of girls and African American and Hispanic learners (Ericson & Guzdial, 2014; Margolis, 2008; Margolis & Fisher, 2003; Zweben & Bizot, 2015). Efforts to increase enrollment among historically underrepresented minorities have taken a number of forms, including national curricular efforts, the creation of new programming environments that emphasize creativity and collaboration, and a wide array of out-of-school programs oriented toward engaging a diverse set of learners. For example, the Exploring Computer Science (ECS) curriculum was designed to broaden participation in computer science by emphasizing aspects of computing such as web design and data analysis and foregrounding the human and social aspects of the domain through culturally relevant curricula and equity-oriented projects (Ryoo et al., 2013). Another approach to introducing a broad range of learners to foundational computer science ideas is to integrate them into existing courses (Barr & Stephenson, 2011; Weintrop et al., 2016).

There are also a growing number of learning environments, technologies, and programs being designed to broaden participation in computing. Low-threshold programming environments have emerged, such as Scratch (Resnick et al., 2009) and Alice (Cooper, Dann, & Pausch, 2000), which use block-based programming interfaces to make it easier for novices to program with little or no prior experience. In these types of environments, programming is framed as a creative activity, allowing learners to create games and interactive stories that can easily be shared with others. Such efforts attract diverse learners that are historically underrepresented in computing (Kelleher, Pausch, & Kiesler, 2007; Maloney et al., 2008). Likewise, tangible computing has emerged as another pathway into computing that can reach audiences historically underrepresented in the field (Brady et al., 2016; Buechley & Hill, 2010).

Block-based programming

The last decade has seen a proliferation of applications utilizing blocks-based programming (Figure 2). Block-based programming leverages a programming-primitive-as-puzzle-piece metaphor that provides visual cues to the user about how and where commands can be used. Users compose programs in these environments by dragging blocks onto a canvas and snapping them together to form scripts. If two blocks cannot be joined to form a valid syntactic statement, the environment prevents them from snapping together, thus preventing syntax errors but retaining the practice of assembling programs instruction-by-instruction. Along with using block shape to denote use, there are other visual cues to help programmers, including color coding by conceptual use and nesting of blocks to denote scope (Maloney



```
when clicked
repeat 10
    move 10 steps
    say Hello! for 2 secs
    play sound meow
```

```
void run ( ) on class MyScene
do in order
    bug moveAndOrientTo( bug_Red , 2.0
    this frogCatchBug();
    this bugLiftsFrog();
    this bugCarriesFrogAway();
    this message();
```

```
var i=0;
var sum=0;
while( i != 4 ){
    i = 1;
    sum = sum + i ;
    i = i + 1 ;
}
console.log( sum );
```

(a) (b) (c) (d)

Figure 2. Block-based programming languages: (a) LogoBlocks, (b) Scratch, (c) Alice, and (d) Code.org's App Lab.

et al., 2010). Collectively, the features of the block-based programming modality and the environments in which they are situated contribute to learners perceiving block-based tools as easy to use (Weintrop & Wilensky, 2015a).

A growing body of research is investigating how blocks-based programming shapes learners' conceptual understanding of computer science concepts and emerging programming practices. For example, researchers have documented a number of habits learners develop while working in blocks-based tools, such as an emphasis on bottom-up programming where learners focus on using specific blocks (Meerbaum-Salant, Armoni, & Ben-Ari, 2011). Likewise, research is documenting how novices learn with blocks-based tools; identifying developmentally-appropriate content and misconceptions learners may develop in blocks-based languages (Franklin et al., 2017; Grover & Basu, 2017; Seiter & Foreman, 2013). Finally, emerging research on comparative studies between block-based and text-based tools shows block-based programming enables students to complete assignments faster (Price & Barnes, 2015) and score higher on content assessments (Weintrop & Wilensky, 2017).

Representation and learning

Research in the Learning Sciences is revealing the ways that the representational infrastructure used in a domain can impact learning and conceptualization of that domain. diSessa (2000) calls this *material intelligence*, arguing for close ties between the internal cognitive process and the external representations that support them: "we don't always have ideas and then express them in the medium. We have ideas *with* the medium" (diSessa, 2000, p. 116, emphasis in the original). These symbolic systems provide a representational infrastructure upon which knowledge is built and communicated (Kaput, Noss, & Hoyles, 2002). For example, Sherin (2001) investigated the use of conventional algebraic representations as compared to programmatic representations in physics courses and found that different representational forms have different affordances with respect to students learning physics concepts.

Wilensky and Papert (2010) give the name structuration to describe this relationship between the representational infrastructure used within the domain and the understanding that infrastructure enables and promotes. While often assumed to be static, Wilensky and Papert show that the structurations that underpin a discipline can, and sometimes should, change as new technologies and ideas emerge. In their formulation of the idea of structurations, Wilensky and Papert document a number of restructurations, shifts from one representational infrastructure to another, including the move from Roman numerals to Hindu-Arabic numerals (Swetz, 1989), the use of the Logo programming language to serve as the representational system to explore geometry (Abelson & diSessa, 1986), and the use of agent-based modeling to represent various biological, physical, and social systems (Wilensky & Rand, 2014). This work highlights the importance of studying representational systems, as restructurations can profoundly change the learnability, power, and communicability of ideas within a domain.

Methods

Context: The AP Computer Science Principles course and exam

This study took place in classrooms during the inaugural year of the AP CSP course. The year-long course focuses on the big ideas of computing and culminates with a 2-hour, 74-question multiple choice exam. Roughly 20% of a student's overall score on the AP Exam is based on their responses to programming questions. The programming questions on the written exam include a mix of block-based and text-based pseudocode questions of the form presented in Figure 1. The assessment used in this study was administered as part of Code.org's CSP curriculum (http://code.org/csp). Code.org's CSP curriculum is a full-year course that introduces high school students to the foundations of modern computing and prepares them for the AP CSP Exam. The course employs curricular and pedagogical strategies that promote equitable teaching practices to support both new-to-computer-science students *and* teachers. Two of the five units in the curriculum involve programming in the JavaScript language through

Code.org's App Lab environment (Figure 2d, http://code.org/applab), which allows students to construct programs in *both* block-based and text-based modalities and includes various affordances to help novice programmers.

The AP pseudocode and our assessment

As shown in Figure 1, the pseudocode created for the AP CSP exam draws inspiration from professional programming languages but includes features and keywords that make it distinct from any existing language. The pseudocode includes keywords for the following topics: Assignment, Display, and Input; Arithmetic Operators and Numeric Procedures; Relational and Boolean Operators; Selection; Iteration; List Operations; Procedures; and Robot commands. The block-based and text-based versions of the pseudocode are the same with a number of exceptions, notably, in the block-based form expressions are wrapped in a rounded-edge rectangle and scope is denoted with nested rectangles, replacing the `{}`s used in the text form. The language consists of 23 keywords including looping constructs (e.g. `FOR`, `REPEAT`), conditional operators (e.g. `IF`, `ELSE`), and list functions (e.g. `REMOVE`, `LENGTH`).

The assessment used in this study asked questions using the block-based and text-based form of the AP CSP pseudocode and followed the design of the Commutative Assessment (Weintrop & Wilensky, 2015b). Each question on the assessment began with a short program presented in either the text-version or block version of the AP CSP pseudocode (e.g. Figure 1) and then followed by the question: "What will the output of the program be?" The assessment was comprised of 20 multiple-choice questions covering five programming topics: variables, loops, conditionals, functions, and program comprehension. For each of the five topics, the assessment asked two questions in the block-based pseudocode form and two in the text-based pseudocode. This counter-balance design ensures that every student answered at least one question for each concept in both forms of the pseudocode.

Data collection and participants

The AP pseudocode assessment was included at the end of the App Development module of the Code.org AP CSP curriculum under the label: AP Pseudocode Practice Questions. The assessment was a supplement to the official course materials. Teachers were notified of its existence via a monthly newsletter that was sent to Code.org users that identified as having an active section of the course. The Code.org curriculum is run through a content management system that tracks individual student as well as classroom progress through the curriculum. This platform was used to administer the assessment and to collect and store student responses. At the beginning of the course, students are asked to create a profile, which includes optionally self-reporting their gender, age, and race. The responses collected by Code.org were de-identified and then shared with researchers. For each student, the data included a full set of responses to the survey questions along with birth year, gender, and race.

The dataset for this project consists of 5,427 students and over 105,000 individual questions responses. The sample was comprised of 1,218 (22.4%) female students, 3,198 (58.9%) male students, and 1,011 students (18.6%) who chose to not provide gender information. Of the 5,427 students, 1,040 (19.2%) learners self-identified as being from an underrepresented minority in computing (URM), while 2,199 (40.5%) of students were classified as not URM and 2,188 (40.3%) of student did not self-report their race. For this work, students that self-identified as Black, Hispanic, LatinX, Native American, or Pacific Islander were categorized as being from a URM. Finally, a majority of participants were between the ages of 15 and 18, which corresponds to the four years of American high-school (9.6% 15 years old; 22.5% 16 years old; 30.7% 17 years old; 27.2 % 18 years old).

Findings

This section presents our analysis of the responses to the AP CSP assessment beginning with overall results then looking at how modality affected learners within racial and gender groups and how outcomes differed by modality across groups. For each calculation, we only include students who answered all of the questions for the calculation being run and provided the necessary demographic data.

Overall findings

Of the 5,427 students who answered at least one question, 4,762 of them that provided responses for all 20 questions on the assessment with a mean score of 16.15 points out of a possible 20 (*SD* 3.9). Of these 4,762 students, 758 students (15.9%) scored a perfect 20 out of 20, while 369 students (7.7%) correctly answered less than half of the questions. The mean correct response rate for girls was 15.79 (*SD* 3.776) while boys average score was 16.10 (*SD* 3.99). An independent samples t-test reveals these two scores to be statistically different from each other $t(3933) = 2.20$, $p = .03$, $d = .08$. Using Cohen's (1992) guidelines, the effect size d (calculated using the pooled standard deviation) is interpreted as a small effect. This result shows boys outperforming girls on the assessment and that the difference in scores between the two groups was statistically significant but relatively small.

Shifting our analysis to students who self-reported as being from an underrepresented minority, we find a similar pattern, but a larger effect size. The mean correct response rate for URM students was 15.26 (*SD* 4.17) while those reporting as not URM scored an average of 16.64 out of 20 (*SD* 3.49). An independent samples t-test shows these two groups to be statistically significantly different, $t(2878) = 9.2, p < .001, d = .36$. This is interpreted as a medium effect. This finding shows URM students scoring significantly lower than non-URM peers, which, along with the previous finding of boys outperforming girls, matches prior work related to achievement and race and gender.

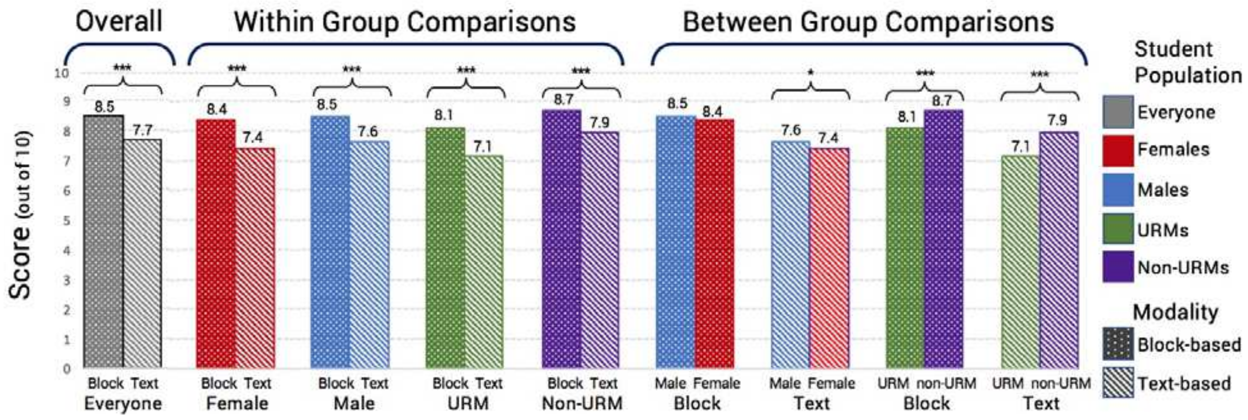


Figure 3. Average scores on the assessment presented as comparisons both within and between population groups.

Differences by modality, within population group

The first research question we pursue in this work investigates how modality affects learners' ability to correctly answer questions on a written computer science assessment. The assessment we designed included 10 block-based questions and 10 text-based questions, providing a dataset that allows us to compare performance between modalities within subjects and within groups. Overall, students scored an average of 8.5 out of 10 on block-based questions (*SD* 1.9) and 7.7 out of 10 on text-based questions (*SD* 2.2), a difference that is statistically significant $t(4761) = 38.14, p < .001, d = .40$ (left-most column of Figure 3). This means students performed better on the block-based question than the text-based questions, a finding that matches prior research looking at students ability to answer written questions by modality (Weintrop & Wilensky, 2015b). However, this result is slightly surprising as the block-based format used in this assessment (Figure 1) lacks many of the features the literature has identified as making block-based tools easier to comprehend, a fact that we will return to later in the discussion.

We now break the population down to see if this trend is consistent across subgroups. Looking at gender, we find that both male and female students perform significantly better on block-based questions: Female $t(1081) = 20.25, p < .001, d = 0.62$ and Male $t(2852) = 29.344, p < .001, d = 0.55$ (columns 2 and 3 in Figure 3 respectively). In the case of female students, this resulted in a .92 point improvement on average while males experienced a .85 increase in mean score. For both genders, there was a medium effect size for performance by modality. Shifting analytic focus to race, we see a similar pattern with both URM and non-URM students performing significantly better on block-based questions versus text-based questions: URM students $t(878) = 18.20, p < .001, d = 0.61$ and non-URM students $t(2000) = 23.74, p < .001, d = 0.53$ (columns 4 and 5 in Figure 3 respectively). For participants that self-identified as a URM, there was, on average a full point difference between mean block-based score and mean text-based score. Non-URM students saw on average score improvement of .76 points between block-based and text-based questions. Like with gender, both statistical differences are found to have a medium effect size. Taken together, **this analysis shows all students perform better on block-based questions, however, looking at the difference in effect size, students from historically underrepresented populations see a greater benefit when questions are asked in the block-based form**. This suggests that using block-based programming is a useful approach to supporting learners from underrepresented groups with the goal of broadening participation in computing. Having looked at differences within populations, we now shift our focus to a comparison across populations.

Differences within modality, across population groups

The previous analysis established that there are differences in performance patterns between block-based and text-based questions. In this section, we conduct a similar analysis, this time looking comparatively across groups to try and identify the sources of differences identified above. We begin by looking at differences by gender. Comparing student performance on block-based questions by gender using an independent sample t-test shows a **not** statistically significant difference between genders: $t(3969) = 1.6$, $p = .12$ (column 6 in Figure 3). If we run the same comparison looking only at text-based questions, we do find a statistically significant difference between gender: $t(3981) = 2.2$, $p = .03$, $d = .08$ (column 7 in Figure 3). Taken together, these findings show that the source of difference in performance by gender reported above is due to results from the text-based questions. In other words, **there is no difference in score by gender on the block-based questions, but a significant difference in scores on the text-based questions**. This suggests that one potential way to shrink the gender gap would be to only use the block-based modality.

If we look at comparative differences by modality between URM and non-URM students, we find a different pattern. The average score for URM students on text-based questions was 7.1 out of 10 (SD 1.4) compared to 7.9 (SD 2.0) for non-URM students with a similar difference in average scores being found for block-based questions: URM students averaged 8.1 out of 10 (SD 2.2) and non-URM students scored an average of 8.7 out of 10 (SD 1.8) (the two right-most columns in Figure 3). For both block-based and text-based questions, these differences are statistically significant: block-based questions $t(2909) = 7.95$, $p < .001$, $d = .30$ and text-based questions $t(2917) = 9.88$, $p < .001$, $d = .38$. The difference in patterns with these results compared to the gender differences observed in the previous paragraph suggests we would not expect the same erosion of performance difference between racial groups if assessments shifted to use the block-based modality exclusively, an idea we further explore below.

Discussion

Potential explanations for these differences

One of the surprising findings from this research is the presence of significant differences between block-based and text-based questions despite the block-based pseudocode not having many of the features that learners have identified as being most helpful. Prior research shows that students identify things such as ease of composition, meaningful shapes and colors, natural language expressions, and ease of browsability as key features of block-based programming tools that helped them learn to program (Weintrop & Wilensky, 2015a). However, none of those features are present in the block-based pseudocode used in this assessment. This suggests there are other factors at play that can explain the observed differences. One potential option is that the block-based visual cues that are present (like the nesting of scope) does play a role in helping learners comprehend the program. This supports prior research that has found navigating scope within a program to be one strength of block-based programming environments (Weintrop & Holbert, 2017), as well as work showing students struggle with unfamiliar syntax (Stefik & Siebert, 2013).

Another possible explanation has less to do with the specifics of the text-based syntax or the rendering of the block-based interface, but instead, has more to do with the affective and perceptual aspects of the questions being asked. Students who are intimidated by text-based programming, or learners who have spent more time working in block-based tools, may feel more comfortable trying to decipher programs written in the block-based modality. In this way, it is less the specifics of the representation that are contributing to these results, but instead, the larger cultural or social dimensions of the representations. Unfortunately, this study does not have the data sources necessary to verify this explanation, as will be discussed later in the section on limitations and future work.

A third potential explanation links the results on the two forms of the pseudocode questions with the programming modality used in class. It is conceivable that classes explicitly designed to broaden participation and have greater numbers of female and/or URM students were more likely to have used block-based tools. At the same time, schools in which male and non-URM students have historically enrolled in computer science may have used this new course as preparation for later computer science instruction and relied on text-based tools. In other words, the results presented above are due to differences in what populations were using what type of programming environment. While we do not have the data from this study to refute this explanation, prior work has not found strong coupling between the modality used for instruction and student performance (Weintrop & Wilensky 2015a, 2017).

Implications of these findings

There are a number of implications that flow directly from the findings presented above. Taken collectively, the presented findings suggest that block-based programming can play a productive role in supporting learners from historically underrepresented populations. This conclusion stems from the finding that female and URM students particularly benefit from the use of block-based questions. Thus, we would expect that if the exam were administered using only block-based questions, the gender gap would shrink, resulting in female students ending the class with a higher grade and potentially increasing the likelihood that they sign up for future computer science courses. There is still much work to be done before we can prove that the transition to block-based assessments produces these outcomes, but the data from this study suggest it is a promising avenue for broadening participation.

A second potential implication of these findings relates to the endpoint of computer science instruction. Historically, computer science education instruction in the United States has had the built-in assumption that text-based programming using a professional language (e.g. Java) was an essential component of early computer science instruction. With the emergence of block-based tools, and other non-professional programming contexts in which authentic computer science can take place, this text-based programming endpoint of computer science instruction is beginning to be challenged. Can someone become proficient in computer science without learning text-based programming? With the introduction of the AP CSP course in the United States, the answer is starting to be yes, but there is still work to be done to completely realize this new endpoint of computing education. The decision to teach an AP course and assess student understanding of computer science entirely in block-based language is a towards new computer science endpoints distinct from text-based programming in professional languages.

Limitations and future work

It is important to note the limitations of this work. For example, while we know some basic information about the participants, like gender and age, there is much we don't know, such as prior programming experience, which would impact the results presented above. Likewise, we know very little about students' classroom experience (e.g. the curriculum followed, the programming tool(s) used, and teacher's prior computing education experience). All of these factors influence the data presented in this study and thus constrain the claims that can be made. A second limitation is that we only have a single, quantitative data source. Ideally, we would have also observed these classrooms to better understand the nature of the instruction as well as interviewed students after taking the survey to gain further insight into how students are making sense of the role the representations in the assessment. We are in the process of designing a second iteration of this study where additional contextual data will be collected to address this limitation.

Conclusion

The push to broaden participation in computing has resulted in numerous computing education initiatives, often championed and implemented with little or no prior research being conducted on the key elements of the program. In the case of the new AP CSP course, the decision to support a plurality of programming tools was informed by research, but the novelty in how the administering body chose to assess student learning led to uncertainty with respect to whether or not the final exam would help address issues of equity and underrepresentation. The analysis presented in this work found that the use of block-based pseudocode programming questions supported learners from historically underrepresented populations, suggesting that this evaluation choice may help the course achieve its stated goals of broadening participation in computing. This analysis advances our understanding of the role that representation can play in supporting diverse students in succeeding in computer science and contributes to the growing knowledge-based of pedagogical strategies and design techniques that can help all students succeed in computing.

References

Abelson, H, & diSessa, A. (1986). *Turtle geometry: The computer as a medium for exploring mathematics*. MIT Press.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? *ACM Inroads*, *2*(1), 48–54.

Brady, C., Orton, K., Weintrop, D., Anton, G., Rodriguez, S., & Wilensky, U. (2016). All Roads Lead to Computing: Making, Participatory Simulations, and Social Computing as Pathways to Computer Science. *IEEE Transactions on Education*, *60*(99), 1–8. https://doi.org/10.1109/TE.2016.2622680

Buechley, L., & Hill, B. M. (2010). LilyPad in the wild: how hardware's long tail is supporting new engineering and design communities. In *Proc. of the 8th ACM Conference on Designing Interactive Systems* (pp. 199–207).

Cohen, J. (1992). A power primer. *Psychological Bulletin*, *112*(1), 155.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, *15*(5), 107–116.

diSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge, MA: MIT Press.

Ericson, B., & Guzdial, M. (2014). Measuring Demographics and Performance in Computer Science Education at a Nationwide Scale. In *Proc. of the 45th ACM SIGCSE Conference* (pp. 217–222). New York, NY, USA: ACM.

Franklin, D., Skifstad, G., Rolock, R., Mehrotra, I., Ding, V., Hansen, A., Weintrop, D. & Harlow, D. (2017). Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum. In *Proc. of the 2017 ACM SIGCSE Conference* (pp. 231–236). New York, NY, USA: ACM.

Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In *Proc. of the 2017 ACM SIGCSE Conference* (pp. 267–272). New York, NY: ACM Press.

Kaput, J., Noss, R., & Hoyles, C. (2002). Developing new notations for a learnable mathematics in the computational era. *Handbook of International Research in Mathematics Education*, 51–75.

Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling alice motivates middle school girls to learn computer programming. In *Proc. of the SIGCHI conference on Human factors in computing systems* (pp. 1455–1464).

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with Scratch. *ACM SIGCSE Bulletin*, *40*(1), 367–371.

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.

Margolis, J. (2008). *Stuck in the shallow end: Education, race, and computing*. The MIT Press.

Margolis, J., & Fisher, A. (2003). *Unlocking the clubhouse: Women in computing*. The MIT Press.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. In *Proc. of the 16th ITiCSE Conference* (pp. 168–172). Darmstadt, Germany: ACM.

Price, T. W., & Barnes, T. (2015). Comparing Textual and Block Interfaces in a Novice Programming Environment (pp. 91–99). Presented at ICER ’15, ACM Press. https://doi.org/10.1145/2787622.2787712

Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., … Silver, J. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60.

Ryoo, J. J., Margolis, J., Lee, C. H., Sandoval, C. D., & Goode, J. (2013). Democratizing computer science knowledge: transforming the face of computer science through public high school education. *Learning, Media and Technology*, *38*(2), 161–181. https://doi.org/10.1080/17439884.2013.756514

Seiter, L., & Foreman, B. (2013). Modeling the Learning Progressions of Computational Thinking of Primary Grade Students. In *Proc. of the 9th Annual ACM ICER Conference* (pp. 59–66). New York, NY,: ACM.

Sherin, B. L. (2001). A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning*, *6*(1), 1–61.

Stefik, A., & Siebert, S. (2013). An Empirical Investigation into Programming Language Syntax. *ACM Transactions on Computing Education*, *13*(4), 1–40. https://doi.org/10.1145/2534973

Swetz, F. (1989). *Capitalism and arithmetic: The new math of the 15th century*. La Salle, Illinois: Open Court.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/10.1007/s10956-015-9581-5

Weintrop, D., & Holbert, N. (2017). From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment. In *Proc. of the 2017 ACM SIGCSE Conference* (pp. 633–638). New York, NY: ACM.

Weintrop, D., & Wilensky, U. (2015a). To Block or Not to Block, That is the Question: Students’ Perceptions of Blocks-based Programming. In *Proc. of the 14th Int. IDC Conference* (pp. 199–208). New York: ACM.

Weintrop, D., & Wilensky, U. (2015b). Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. In *Proc. of the 11th ICER Conference*. New York, NY: ACM.

Weintrop, D., & Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 3.

Wilensky, U., & Papert, S. (2010). Restructurations: Reformulating knowledge disciplines through new representational forms. In J. Clayson & I. Kallas (Eds.), *Proc. of Constructionism 2010*. Paris, France.

Wilensky, U., & Rand, W. (2014). *Introduction to Agent-based Modeling*. Cambridge, MA: MIT Press.

Zweben, S., & Bizot, B. (2015). 2014 Taulbee Survey. *COMPUTING*, *27*(5).