DOI:10.1145/3341221

David Weintrop

▸ **Mark Guzdial,** Column Editor

Education

Block-based Programming in Computer Science Education

Considering how block-based programming environments and tools might be used at the introductory level and beyond.

BLOCK-BASED PROGRAMMING IS increasingly the way that learners are being introduced to the practice of programming and the field of computer science more broadly. Led by the success of environments like Scratch (see the figure appearing later in this column) and initiatives like Code.org's Hour of Code, block-based programming is now an established part of the computer science education landscape. While not a recent innovation (for example, LogoBlocks has been around since the mid-1990s), the last decade has seen a blossoming of new toys, games, programming environments, and curricula that incorporate block-based programming features. Given this growing presence, it is important that we as a community look critically at the block-based programming modality to understand its affordances, drawbacks, and identify how best to use it as a means to welcome people into the discipline of computer science and support them as they grow and learn.

What Is Block-based Programming?

Block-based programming has a number of key features that make it distinct from conventional text-based programming and other visual programming approaches. Block-based programming uses a programming-primitive-as-puzzle-piece metaphor as a means of providing visual cues to the user as to how and where commands may be used. Figure 1b shows a block-based program written in Scratch. Block-based programming environments have been designed for children as young as five years old but most environments are designed for kids ages eight to 16. Writing a program in a block-based environment takes the form of dragging-and-dropping programming instructions together. If two instructions cannot be joined to produce a valid statement, then the environment prevents them from snapping together. In this way, block-based programming environments can prevent syntax errors while still retaining the practice of authoring programs by assembling statements one-by-one.

While the visual cues and mitigation of syntax errors are key ingredients in supporting novices in having early programming success, there are additional features of block-based programming that support beginners. For example, block-based programming environments present the available set of commands to the user in the form of an easily browsed blocks palette from which the user can drag the command into their program (left-hand side of Figure 1a). Within the palette, the blocks

Students working on Scratch projects during a summer algebra camp in Irvine, CA, USA.

PHOTO BY JEFF GRITCHEN/DIGITAL FIRST MEDIA/ORANGE COUNTY REGISTER VIA GETTY IMAGES

are conceptually organized and color coded. This allows users to browse the set of available commands to see what is possible rather than needing to know before-hand what can be done in the language. At the same time, the drag-and-drop composition approach removes the challenge of typing and finding uncommon punctuation marks on the keyboard, making programming more accessible for people who struggle with typing. Another notable feature of block-based programming environments is that the graphical presentation of each programming statement makes it possible to use natural language to describe the behavior of the command. For example, incrementing the value of a variable, which in a programming language like Java would look like this: `x=x+1;`, can be accomplished with a command that reads: change `x` by `1`. Given the myriad of supports present in block-based environments, it is important to understand if, how, and why this approach is an effective way to introduce novices to programming.

The Case for Block-based Programming

A good place to start the discussion of the benefits of block-based programming is with the most successful (to date) block-based programming environment: Scratch.[2] The goal of Scratch was to create a programming environment with sufficient scaffolds for novices to start to program with little or no formal instruction (low threshold) while also being able to support sophisticated programs (high ceiling). At the same time, it was important that the environment support a variety of types of programmers and programs (wide walls) and provide a means for programmers to share the programs they authored and participate in a larger community of programmers. Since its launch, over 35 million users have created accounts on the Scratch website and almost 40 million projects have been shared with a majority of users being under the age of 14. Further, research has shown block-based tools like Scratch and Looking Glass Alice can be an effective environment for welcoming learners from populations historically underrepresented in computing fields.[1] These numbers reinforce the idea that block-based programming is playing a significant and important role in introducing youth to programming.

To understand how learners make sense of the block-based modality and understand the scaffolds that novice programmers find useful, I conducted a series of studies in high-school computer science classrooms. As part of this work, I observed novices writing programs in block-based tools and interviewed them about the experience. Through these interviews and a series of surveys, a picture emerged of what the learners themselves identified as being useful about the block-based approach to programming. Students cited features discussed here such as the shape and visual layout of blocks, the ability to browse available commands, and the ease of the drag-and-drop composition interaction. They also cited the language of the blocks themselves, with one student saying

"Java is not in English it's in Java language, and the blocks are in English, it's easier to understand." I also surveyed students after working in both block-based and text-based programming environment and they overwhelmingly reported block-based tools as being easier.[5] These findings show that students themselves see block-based tools as useful and shed light as to why this is the case.

To investigate learning outcomes associated with block-based programming, I conducted a quasi-experimental study in two high school computer science classrooms. The two classrooms used the same programming environment with one difference: one environment presented the code in a block-based interface while the other had a text-based interface. The underlying programming language was the same between the two meaning anything that could be done in one modality could also be done in the other. Starting on the first day of school, the two classes spent five weeks working through the same curriculum and were taught by the same teacher. The study was designed to control for as many factors as possible aside from the programming modality. After the five-week introduction, students in the block-based condition scored significantly high on content assessments than their text-based peers.[6]

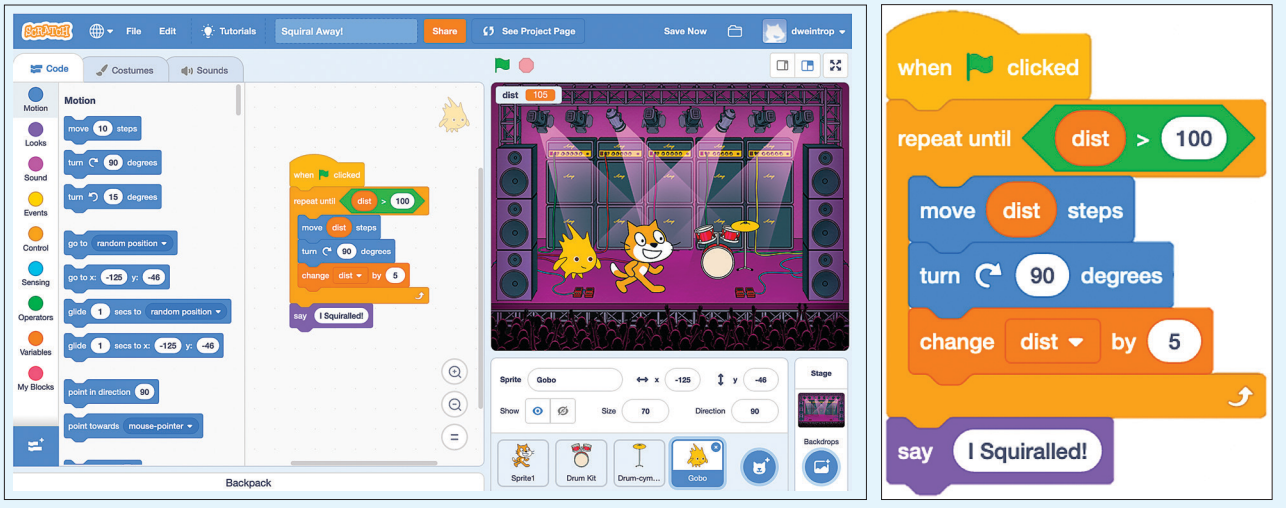
It is worth thinking about what role block-based languages might play in the design of computational tools.

Challenges and Open Questions

While research has shown the potential of block-based environments, there are still challenges and open questions related to the role of block-based programming in introductory computing contexts. One significant question relates to perceptions of block-based tools and whether or not dragging-and-dropping colorful and playful programming commands constitutes "real programming." While an experienced programmer may be able to see the conceptual equivalence between the repeat block in Scratch and a for loop in Java, the same is not necessarily true for beginners. As part of my classroom interviews, some students expressed concerns over the authenticity of block-based programming. For example, one student stated: "*if we actually want to program something, we wouldn't have blocks*," which calls into question the potential utility of block-based tools. Students also expressed concern over the expressive power of block-based environments, saying things like "*blocks are limiting, like you can't do everything you can with Java, I guess. There is not a block for everything.*" While this statement is not necessarily true of all block-based environments (for example, there are numerous block-based interfaces for Java), the fact that students perceive this difference is a challenge educators face.[5]

A second open question surrounding block-based programming is whether or not block-based tools help learners with transitioning to text-based languages. There have been some documented examples of students learning concepts in block-based environments and successfully transferring those ideas to a text-based

The Scratch programming environment (a) and a block-based program written in Scratch (b).*



(a) (b)

a In early programming work with the Logo language (from which Scratch is based), students would draw square spirals that came to be called squirals: the image (and program) reflects that history.

language, however, my own research has not replicated these results. In a continuation of the quasi-experimental classroom study discussed above, after students finished their five-week introduction to programming in the block-based and text-based introductory environments, all students transitioned to the Java programming language. After 10 weeks of learning Java, I readministered the content assessment. Despite students in the block-based condition scoring significantly higher after the introductory portion of the course, there was no significant difference in scores between students in the two conditions. This means the gains associated with the block-based introduction did not translate to students being further ahead when learning Java but also did not hamper their transition.[3] Understanding how to better scaffold learners moving between modalities, and the role of the teacher in this processes is a direction of future work for the field.

What the Future Block-based Programming Might Be

So, what is next for block-based programming? First, as the research described in this column suggests, the literature shows block-based programming should have a home in computer science education. One version of that is in the role it is currently playing, that of introductory tools designed to welcome novices to the field, either in upper elementary grades (ages 10 to 14) or high-school (up to age 18). As to what exactly that looks like and how such environments can support learners in moving beyond block-based tools is an open question. One potential direction is hybrid and bidirectional programming environments that blend block-based and text-based tools, giving the learners agency for deciding how and when to switch programming representation. This is one active and exciting area of design research in the area of introductory computing.

Looking beyond introductory contexts, there is a larger question about the potential role of block-based tools in the world of computer science. Currently, there is an assumption that block-based tools serve as an entry point with the expectation that learners move beyond it to conventional text-based programming languages. However, as the Computer Science for All movement progresses and programming becomes a more universal literacy, it is worth thinking about what role block-based languages might play in the design of computational tools. If it is possible to do significant, non-trivial tasks in block-based environments, should we still expect all learners, even those not likely to pursue a degree in computer science, to learn text-based programming? For example, we created a block-based interface for controlling industrial robots and found it be easier for adult novices to use than existing robotics programming environment.[4] Given the success of this design, it becomes easy to imagine a world with countless domain-specific block-based programming tools that put the power of computing at the fingertips of those who are proficient with block-based programming. This is not to say this is what the future holds but instead I put this forward as a way to think about new possible end-points for computing education and a more expansive view of the potential of block-based programming in the technological world that awaits. C

References

1. Kelleher, C., Pausch, R., and Kiesler, S. Storytelling Alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2007), 1455–1464.
2. Resnick, M. et al. Scratch: Programming for all. *Commun. ACM 52*, 11 (Nov. 2009), 60.
3. Weintrop, D. Modality Matters: Understanding the Effects of Programming Language Representation in High School Computer Science Classrooms (Ph.D. Dissertation). Northwestern University, Evanston, IL, 2016.
4. Weintrop, D. et al. Evaluating CoBlox: A comparative study of robotics programming environments for adult novices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems 366*, (2018), 1–12; https://doi.org/10.1145/3173574.3173940
5. Weintrop, D. and Wilensky, U. To block or not to block, that is the question: Students' perceptions of blocks-based programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (2015), 199–208; https://doi.org/10.1145/2771839.2771860
6. Weintrop, D. and Wilensky, U. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE), 18*, 1 (2017), 3; https://doi.org/10.1145/3089799

David Weintrop (weintrop@umd.edu) is Assistant Professor in the College of Education and the College of Information Studies at the University of Maryland, College Park, MD, USA. .