

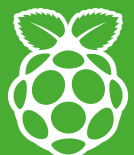
Section 3: Computing topics

The role of block-based programming in computer science education

David Weintrop (University of Maryland, USA)

Weintrop, D.(2021). The role of block-based programming in computer science education. In Understanding computing education (Vol 1). Proceedings of the Raspberry Pi Foundation Research Seminar series.

Available at: rpf.io/seminar-proceedings-2020



Raspberry Pi

Section 3: Computing topics

The role of block-based programming in computer science education

David Weintrop (University of Maryland, USA)

Abstract

Block-based programming environments are increasingly becoming the way that young learners are being introduced to the practice of programming and the field of computer science more broadly. Environments such as Scratch, MIT AppInventor, Code.org's AppLab, and block-based interfaces for physical devices provide inviting and accessible pathways into the world of programming. In this article, I share findings from a series of studies investigating the use of block-based programming in K-12 classrooms. In particular, this research compares block-based programming to conventional text-based programming languages and explores the transition from introductory block-based tools to professional programming languages. The results of the study found that high school students score better on tests after learning to program in a block-based tool compared to peers who learned with a text-based language. The study also found that after transitioning to a professional text-based programming language (Java), there was no difference in programming performance in terms of scores on a content assessment or differences in programming practices employed. The implications of these findings suggest that block-based programming is an effective way to introduce learners to programming but open questions remain about how to best integrate it into formal classroom instruction.

Introduction

Led by the popularity of environments like Scratch, MIT AppInventor, and the growing ecosystem of programming environments built with the Blockly library, block-based programming is increasingly becoming the way that learners are being introduced to the practice of programming and the field of computer science more broadly (Bau et al., 2017; Resnick et al., 2009; Weintrop, 2019). Along with virtual programming environments, a growing number of physical devices support block-based programming, including Sphero, BBC micro:bit, Lego Mindstorms, and several block-based programming environments for the Raspberry Pi family of microprocessors. While not a recent innovation (block-based environment first emerged in the mid-1990s), the last decade has seen a blossoming of block-based programming environments and computing curricula that rely upon block-based tools. A recent review of the academic literature identified 99 unique block-based programming environments (Lin & Weintrop, 2021). This has, in turn, led to a growing body of research seeking to understand the affordance of block-based tools and articulate their role in computer science education (Franklin et al., 2017; Grover & Basu, 2017; Price & Barnes, 2015; Weintrop, Hansen, et al., 2018). As block-based tools become more widespread, it is important that we as educators understand the affordances and drawbacks of these environments so we are best able to

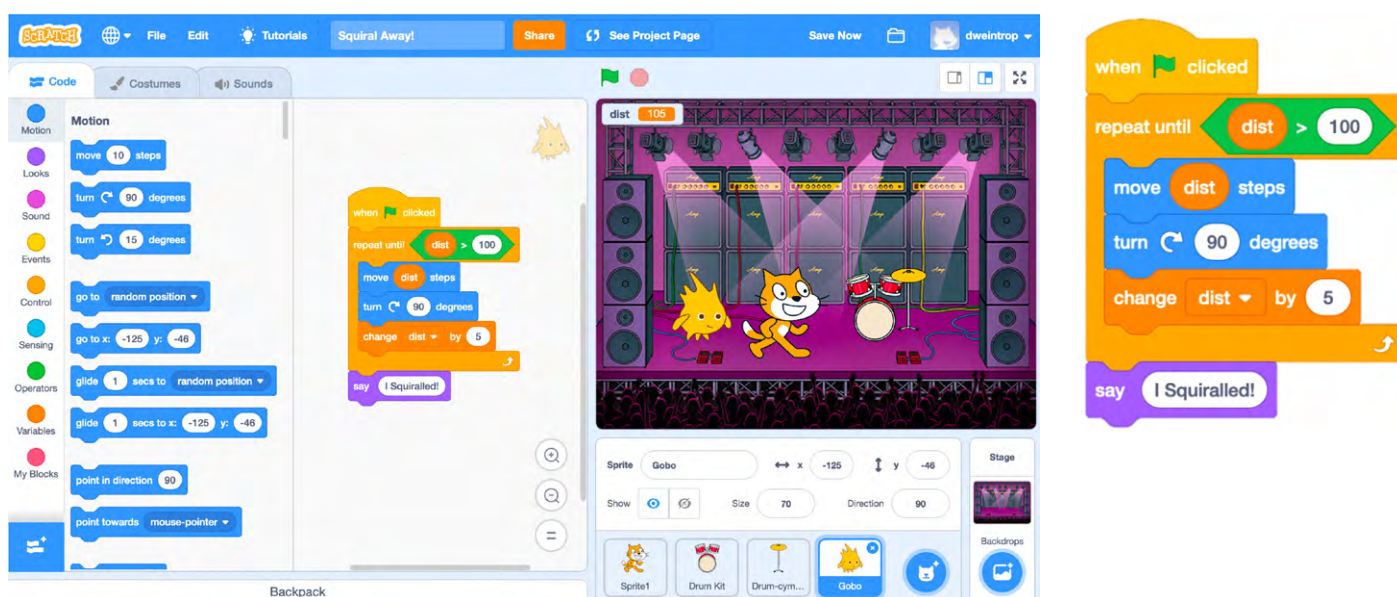


Figure 1. The Scratch programming environment (left) and a block-based program written in Scratch (right)

support learners early in their computer science careers.

The goal of this article is to present findings from a series of research studies seeking to understand the impact of using block-based programming environments in classrooms. In particular, we pursue questions seeking to understand how block-based instruction compares to text-based instruction and to understand if and how the experience of learning to program in a block-based environment better prepares learners for future text-based programming. In doing so, this work seeks to elucidate the potential role of block-based programming in formal education and equip educators to effectively use block-based programming as part of their instruction.

What is block-based programming?

Block-based programming is a graphical approach to programming that uses a

programming-command-as-puzzle-piece metaphor to visually convey information about the programming commands available to the user and how they can be used (Figure 1). Through the inclusion of visual, organisational, and audio cues, block-based programming environments can help novices write functioning programs from the start. The defining feature of block-based programming environments, and the source of their name, is that programming commands are presented as blocks where the shape of the block defines how and where it can be used (Maloney et al., 2010). To assemble a program, the user drags blocks onto the canvas (the area where the program is written) and snaps the blocks together, often accompanied by an audible click. Only valid combinations of blocks can be snapped together, in this way, block-based programming environments can prevent syntax errors by not allowing for invalid programs to be written.

Along with the visual layout of the blocks, there

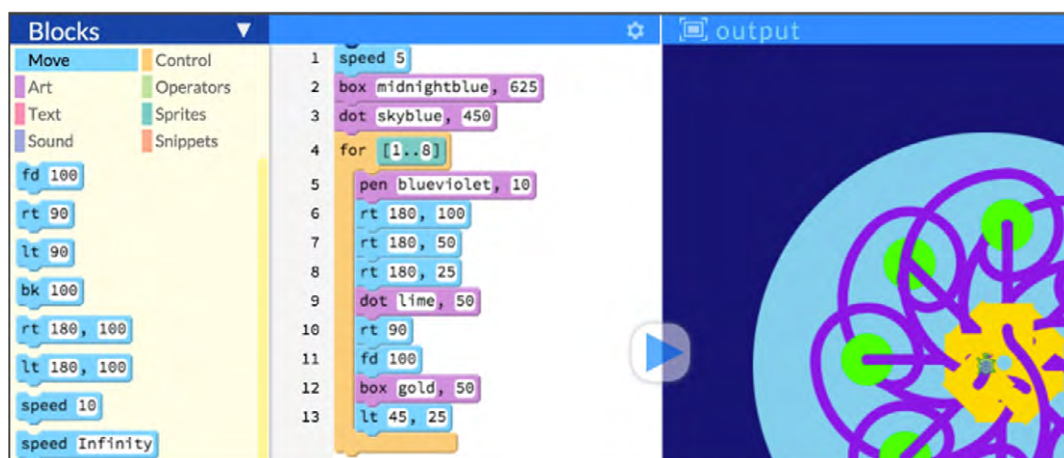
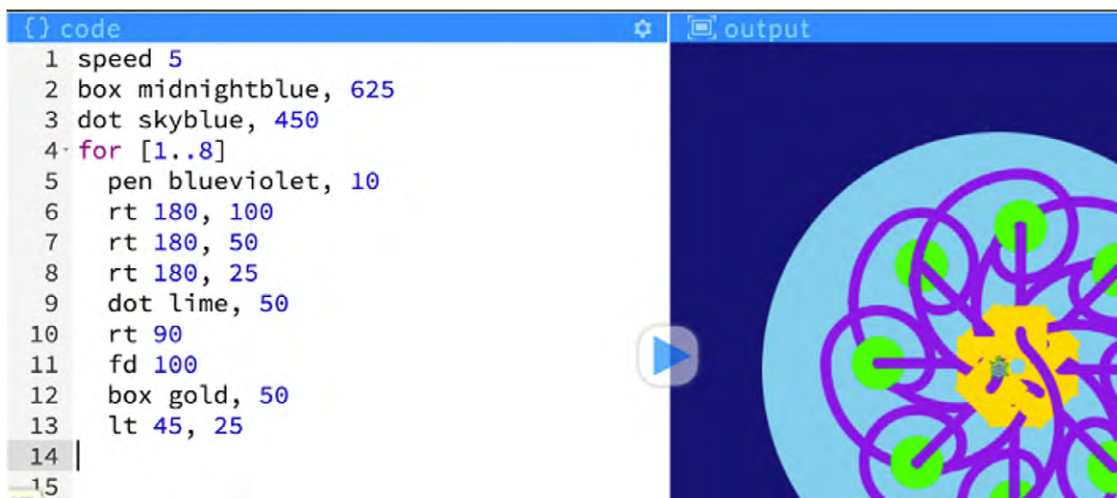


Figure 2. The block-based version of the Pencil.cc programming environment.

are a number of other features that make block-based programming easy for novices with little prior programming experience. For example, as part of a study of high school students learning to program, students talked about how the arrangement of available blocks (left side of Figure 1) made it easy to figure out what was possible in the programming language (Weintrop & Wilensky, 2015a). Students also talked about how the drag-and-drop approach to assembling programs was easier than typing in programming commands one character at a time. This is especially true considering many programming languages require the user to type in uncommon and often mysterious punctuation as part of writing a functioning program. Another feature of block-based language students cited as contributing to their ease-of-use is how the blocks themselves are easier to read when compared to a conventional text-based language. As one student said, *“Java is not in English it’s in Java language, the blocks are in English, it’s easier to understand”*. Collectively, these various affordances lead learners to perceive block-based programming to be easier for novices.

The case for block-based programming

A central and important question about the potential role of block-based programming environments in formal education is whether or not students learn computer science concepts when programming in block-based environments. A related question is how students learn with block-based environments compared to comparable text-based programming languages? In other words, do students learn more in blocks or text? To answer this question, I conducted a quasi-experimental study in two high-school computer science classrooms. Students in one classroom learned using a block-based programming environment (Figure 2) while students in the other classroom used a text-based programming environment (Figure). Importantly, everything about the environments was identical aside from the way programs were presented and authored, including the programming language itself, which was the exact same character-by-character between the two environments. The study began on the first day of school and lasted for five weeks with both classes going through the same curriculum and being taught by the same teacher. As much as



```

() code
1 speed 5
2 box midnightblue, 625
3 dot skyblue, 450
4 for [1..8]
5   pen blueviolet, 10
6   rt 180, 100
7   rt 180, 50
8   rt 180, 25
9   dot lime, 50
10  rt 90
11  fd 100
12  box gold, 50
13  lt 45, 25
14
15
  
```

Figure 3. The text-based version of the Pencil.cc programming environment.

possible, everything was kept constant between the two classrooms aside from the programming environment.

After learning to program in either the block-based or text-based environments, students took a programming assessment where the questions were asked in both block-based and text-based forms (Weintrop & Wilensky, 2015b). At the conclusion of the five-weeks of instruction, students who learned with the block-based environment scored higher on the content assessment than their peers who learned with the text-based environment (Weintrop & Wilensky, 2017a). This finding is important evidence showing block-based programming to be an effective way to introduce novices to programming.

As part of this study, students also took an attitudinal survey to explore their interest in computer science, their confidence with the discipline, and get an overall sense of their feelings about computer science. After working in a block-based environment for five weeks, learners were significantly more confident in their computer science abilities and their interest

in the field had grown (Weintrop & Wilensky, 2017a).

This study was particularly focused on one block-based programming environment (Figures 2 and 3), however, the finding that students perform better in block-based environments has been replicated in other work. For example, through a partnership with code.org, I investigated how students performed on a computer science content assessment that asked questions using pseudocode presented in both block-based and text-based forms (Figure 4). This pseudocode was developed for the Advanced Placement (AP) Computer Science Principles (CSP) exam that is administered to high school students across the United States. The challenge with this assessment is that the organisation that designs and administers the test does not know what programming language students have learned with or if they learned in a block-based or text-based environment. As such, the test must be appropriate for learners who learned to program with block-based environments and learners who learned with text-based languages. The solution to this problem was for the AP CSP test to use a pseudocode

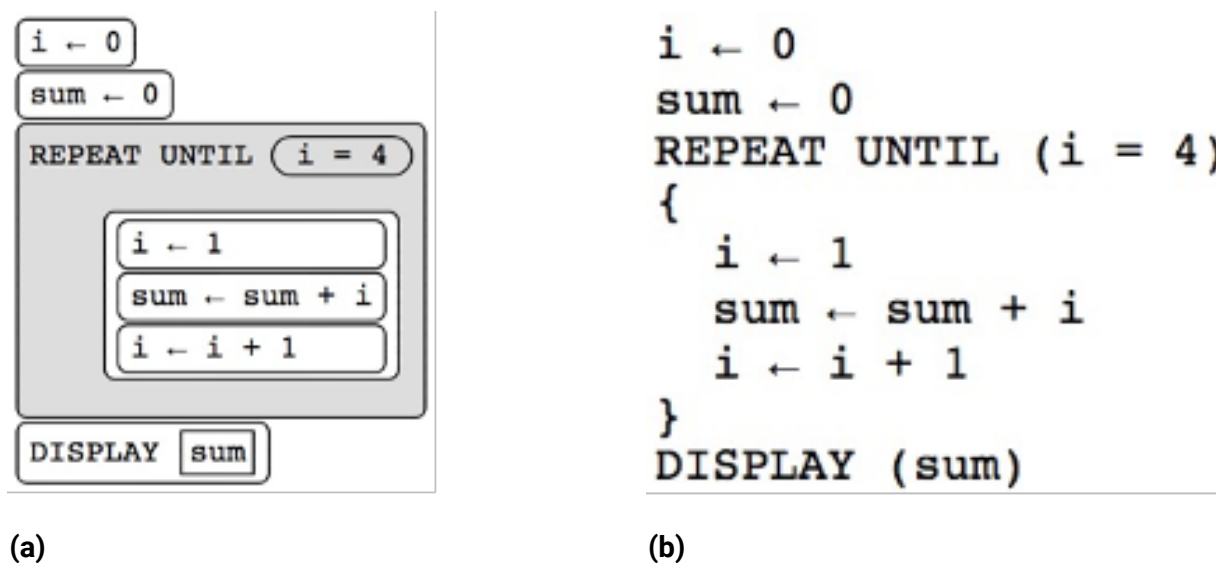


Figure 4. The (a) block-based and (b) text-based pseudocode from the AP Computer Science Principles exam.

with both a block-based form (Figure 4a) and a text-based form (Figure 4b).

An analysis of over 5,000 students from across the United States who took a 20-question content assessment comprised both block-based and text-based questions using AP CSP's pseudocode found that students scored significantly higher on questions asked in the block-based form than questions asked in the text-based form (Weintrop et al., 2019). Further, in breaking down results by race and gender, we found that women and students from racial backgrounds that have been historically excluded in computing saw greater benefits to questions asked in the block-based form (Weintrop & Killen et al., 2018). This finding provides additional evidence for the importance of including block-based programming in formal education, especially as it relates to goals of equity and broadening participation in the field.

Drawbacks and challenges

While the evidence presented above shows

the value of block-based instruction in K-12 classrooms, this work also identified some drawbacks and challenges related to the use of block-based environments in classrooms. In analysing student feedback to identify what students found to be useful about block-based programming, we also found that students identified a series of drawbacks (Weintrop & Wilensky, 2015a). For example, some students expressed concerns related to the authenticity of block-based programming, as one student put it, "if we actually want to program something, we wouldn't have blocks." Other drawbacks mentioned by students included concerns that block-based programming environments were inherently less powerful than text-based programming languages and that writing programs in block-based environments was slower than authoring programs in text-based languages.

A second drawback, or at least an open question, related to block-based programming is if and how block-based programming prepares learners for future computer science instruction

using text-based programming languages. In a continuation of the study discussed above, after five weeks of learning in either a block-based or text-based introductory programming environment, we followed students as they transitioned to instruction in Java. After ten weeks of learning Java, students in both conditions took another content assessment. The result of that assessment showed that there was no difference in performance on the assessment based on their introductory experiences (Weintrop & Wilensky, 2019). That is to say, students scored the same on the assessment after ten weeks of Java instruction regardless of which introductory environment they used, so the gains found after five weeks for students learning in the block-based environment were no longer present. We also found there to be no significant difference in terms of the programming practices employed while authoring programs and that students from both introductory experiences showed similar patterns in the types and frequency of syntax errors encountered (Weintrop & Wilensky, 2018). One important thing to note about this study was that the teacher who taught these classes did not employ any specific pedagogical strategies to help bridge the transition from block-based to text-based programming. In other studies where successful transfer has been documented, there are usually explicit bridging strategies employed by the instructor(s) to help learners make the transition (Dann et al., 2012). Questions related to pedagogy and how best to prepare instructors to teach computer science remains an active area of research (Franklin et al., 2020; Yadav & Berges, 2019).

Implications and recommendations

Implications

The primary implication of this research is that block-based programming has a home in computer science classrooms. However, there

are still open questions that need to be answered in terms of how best to use block-based programming to help support learning, both in the classroom and beyond (Brown et al., 2016). While much work remains to be done to figure out exactly how best to utilise this programming approach, the findings cited above and reported elsewhere show block-based programming to be an effective way to introduce novices to the practice of programming and the field of computer science more broadly.

A second implication from this work stems from the finding that students who learned using a block-based programming environment did not see any significant advantage from that experience compared to their text-based peers after transitioning to a text-based language. The important takeaway from this finding is the idea that transfer does not come for free. That is to say, while there are clear conceptual links between programming in a block-based environment and programming in a text-based language, learners do not necessarily see those links and make the connections on their own. This is a place where pedagogy and the teacher play an essential role. Providing explicit instruction to help learners make the connection between blocks and specific programming keywords can help scaffold that transition and help learners build upon conceptual gains made in block-based tools. While there is some work showing this to be effective (Dann et al., 2012), more work needs to be done to more fully understand how best to support learners in making this transition.

Recommendations

So, at the end of the day, where does that leave us in terms of what is the best way to teach students to program? When I am asked by teachers if they should use a block-based environment or start with a text-based programming language, my response is: why not

both? Up to this point, block-based environments and text-based languages have been presented as mutually exclusive options. However, this need not be the case. There are a growing number of programming environments that blend block-based and text-based features like BlueJ's Frame-based editor (Kölling et al., 2015) and others that support both block-based and text-based programming like Pencil code (Bau et al., 2015). I am increasingly excited about programming environments that support both block-based and text-based programming, where the learners can decide which interface they want to see and can move back and forth between the two forms. I call these dual-modality environments (Weintrop & Wilensky, 2017b) and a growing body of research is showing them to be an effective approach to support novices early in their learning while also providing scaffolds for them to transition from block-based composition to more conventional text-based programming (Blanchard et al., 2020; Matsuzawa et al., 2015; Weintrop & Holbert, 2017).

A second important question to ask when thinking about the role of block-based programming in computer science education, especially as it relates to the transition to text-based instruction, is whether or not that transition is even necessary. Do all students need to learn to program in professional text-based languages? If the goal is to prepare students for a career in computer science, then the answer is probably yes, students would need to learn to program with professional text-based languages. However, it is worth re-examining whether the goal of computer science instruction should be to prepare learners for careers in the field. While that certainly is one desirable endpoint of computer science instruction, it is important to consider alternative endpoints, such as preparing learners for careers outside of computer science, equipping students to be informed technologically-savvy citizens, and empowering learners to pursue their own goals and interests

through computing (Tissenbaum et al., In Press). The idea that block-based programming may be a sufficient endpoint for computer science instruction is also bolstered by the growing number of block-based environments designed for real-world applications such as data sciences (Bart et al., 2017) and industrial robotics programming (Weintrop, Afzal, et al., 2018).

Conclusion

The goal of this article was to share findings from research investigating the role of block-based programming in computer science education. While block-based environments such as Scratch have had a significant impact on youth learning to program in informal environments, the role of block-based programming in formal classroom contexts was less clear. In this article, I presented results that show block-based programming to be an effective way to welcome learners to the field of computer science. At the same time, there are still open questions related to how best to utilize block-based environments as part of formal computer science instruction. In particular, how to address student concerns around questions of authenticity and how to effectively scaffold learners in the transition to text-based languages. In discussing these challenges, I put forward the idea of dual-modality programming environments that support both block-based and text-based forms of authorship as one potential way to address this concern. The work reviewed here, along with the growing body of research around the design of introductory programming environments, curricula, and pedagogy, collectively are poised to lay the groundwork for the infrastructure needed to prepare all learners to succeed in an increasingly computational world.

References

- Bart, A. C., Tibau, J., Kafura, D., Shaffer, C. A., & Tilevich, E. (2017). Design and Evaluation of a Block-based Environment with a Data Science Context. *IEEE Transactions on Emerging Topics in Computing*, *PP(99)*, 1–1.
- Bau, D., Bau, D. A., Dawson, M., & Pickens, C. S. (2015). Pencil Code: Block Code for a Text World. *Proc. of the 14th Int. Conference on Interaction Design and Children*, 445–448.
- Bau, David, Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: Blocks and beyond. *Communications of the ACM*, *60(6)*, 72–80.
- Blanchard, J., Gardner-McCune, C., & Anthony, L. (2020). Dual-Modality Instruction and Learning: A Case Study in CS1. *Proc. of the 51st ACM Technical Symposium on Computer Science Education*, 818–824.
- Brown, N. C. C., Mönig, J., Bau, A., & Weintrop, D. (2016). Future Directions of Block-based Programming. *Proc. of the 47th ACM Technical Symposium on Computing Science Education*, 315–316. Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to Java. *Proc. of the 43rd ACM Technical Symposium on Computer Science Education*, 141–146.
- 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. *Proc. of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 231–236.
- Franklin, D, Coenraad, M., Palmer, J., EATINGER, D., Zipp, A., Anaya, M., White, M., Pham, H., Gökdemir, O., & Weintrop, D. (2020). An Analysis of Use-Modify-Create Pedagogical Approach's Success in Balancing Structure and Student Agency. *Proc. of the 2020 ACM Conference on International Computing Education Research*, 14–24.
- Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. *Proc. of the 2017 ACM Technical Symposium on Computer Science Education*, 267–272.
- Kölling, M., Brown, N. C. C., & Altadmri, A. (2015). Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming. *Proc. of the Workshop in Primary and Secondary Computing Education*, 29–38.
- Lin, Y., & Weintrop, D. (2021). The Current Landscape of Block-based Programming Environments. *Paper Presented at the Annual Meeting of the American Educational Research Association* (2021).
- 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 Trans. on Computing Education*, *10(4)*, 16.
- Matsuzawa, Y., Ohata, T., Sugiura, M., & Sakai, S. (2015). Language Migration in non-CS Introductory Programming through Mutual Language Translation Environment. *Proc. of the 46th ACM Technical Symposium on Computer Science Education*, 185–190.
- Price, T. W., & Barnes, T. (2015). *Comparing Textual and Block Interfaces in a Novice Programming Environment*. 91–99.
- Resnick, M., Silverman, B., Kafai, Y., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., & Silver, J. (2009). Scratch: Programming for all. *Communications of the ACM*, *52(11)*, 60.
- Tissenbaum, M., Weintrop, D., Holbert, N., & Clegg, T. (In Press). The Case for Alternative Endpoints in Computing Education. *British Journal of Educational Technology*.
- Weintrop, D. (2019). Block-based Programming in Computer Science Education. *Commun. ACM*, *62(8)*, 22–25.
- Weintrop, D, Afzal, A., Salac, J., Francis, P, Li, B., Shepherd, D. C., & Franklin, D. (2018). Evaluating CoBloX: A Comparative Study of Robotics Programming Environments for Adult Novices. *Proc. of the 2018 CHI Conference on Human Factors in Computing Systems*, 366:1-12.
- Weintrop, D, Hansen, A. K., Harlow, D. B., & Franklin, D. (2018). Starting from Scratch: Outcomes of Early Computer Science Learning Experiences and Implications for What Comes Next. *Proc. of the 2018 ACM Conference on International Computing Education Research*, 142–150.
- Weintrop, D, & Holbert, N. (2017). From Blocks to Text and Back: Programming Patterns in a Dual-Modality Environment. *Proc. of the 2017 ACM Technical Symposium on Computer Science Education*, 633–638.
- Weintrop, D, Killen, H., & Franke, B. (2018). Blocks or Text? How programming language modality makes a difference in assessing underrepresented populations. *Proc. of the International Conference on the Learning Sciences 2018*, 328–335.
- Weintrop, D, Killen, H., Munzar, T., & Franke, B. (2019). Block-based Comprehension: Exploring and Explaining Student Outcomes from a Read-only Block-based Exam. *Proc. of the 50th ACM Technical Symposium on Computer Science Education*, 1218–1224.
- Weintrop, D, & Wilensky, U. (2017a). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Trans. on Computing Education*, *18(1)*, 3.
- Weintrop, D, & Wilensky, U. (2018). How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *International Journal of Child-Computer Interaction*, *17*, 83–92.
- Weintrop, D, & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers & Education*, *142*, 103646.
- Weintrop, D, & Wilensky, U. (2017b). Between a Block and a Typeface: Designing and Evaluating Hybrid Programming Environments. *Proc. of the 2017 Conference on Interaction Design and Children*, 183–192.
- Weintrop, D, & Wilensky, U. (2015a). To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-based Programming. *Proc. of the 14th International Conference on Interaction Design and Children*, 199–208.
- Weintrop, D, & Wilensky, U. (2015b). Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. *Proc. of the 11th Annual International Computing Education Research Conference*, 101–110.
- Yadav, A., & Berges, M. (2019). Computer Science Pedagogical Content Knowledge: Characterizing Teacher Performance. *ACM Trans. on Computing Education*, *19(3)*, 1–24.



Raspberry Pi

www.raspberrypi.org

 [@raspberrypi](https://www.facebook.com/raspberrypi)

 [@raspberrypi](https://www.instagram.com/raspberrypi)

 [@Raspberry_Pi](https://twitter.com/Raspberry_Pi)

 [raspberrypi](https://www.youtube.com/raspberrypi)