# How the Block-based, Text-based, and Hybrid Block/Text Modalities Shape Conceptual Understandings of Programming Concepts

David Weintrop[1] & Uri Wilensky[2]

weintrop@umd.edu, uri@northwestern.edu

[1]College of Education and College of Information Studies, University of Maryland
[2]Learning Sciences and Computer Science, Northwestern University

*Abstract*
The landscape of introductory programming environments is going through a period of rapid growth, with many new environments emerging and quickly finding their ways into classrooms. While there are many benefits to the release and adoption of new ways of introducing novices to the practice of programming and powerful ideas of computing, our understanding of the cognitive implications of these different approaches has not kept pace. In this paper, we seek to begin to fill in this gap by presenting results from a comparative study investigating how different modalities (block-based, text-based and hybrid block/text) influence learners' emerging conceptual understandings of foundational programming ideas. In doing so, we advance our understanding of the roles of representation and modality in supporting learning.

## Introduction

The last decade has seen a proliferation of introductory programming environments. Led by the popularity of tools like Scratch and Alice, block-based programming is increasingly becoming the modality of choice for programming environments designed for novices of all ages. Further, as the use of block-based tools in formal educational settings grows, so too do the number of tools and strategies designed to help scaffold learners in transitioning from introductory block-based environments to conventional text-based programming languages, including hybrid environments that blend block-based and text-based features. Despite the growth in the variety of introductory programming tools used, relatively little work has been done investigating how these different modalities affect learners' conceptualizations of the foundational computing ideas they are interacting with, such as variables, conditional logic, iterative logic, and functions. This paper seeks to begin to fill that gap in our knowledge, specifically, we seek to answer the following research question:

> *How does a modality (block-based, text-based and hybrid blocks/text) shape learners' emerging understandings of foundational programming concepts?*

To answer this question, we conducted a 5 week, quasi-experimental study in which high school students in an introductory computer science classroom used either a block-based, text-based, or hybrid block/text programming environment. During the study, students worked through a curriculum that covered a number of foundational concepts. At the conclusion of the study, post assessments, surveys, and interviews were conducted to evaluate the conceptual outcomes of the three programming modalities used. This submission presents the results of an analysis investigating how modality shapes emerging understanding, focusing specifically on students' conceptualizations of variables. The full paper will also include an analysis of functions, conditional logic, and iterative logic.

## Prior work

This work builds of two interrelated literatures: work on the role of representation and learning, and design work investigating the creation of intuitive and accessible programming tools.

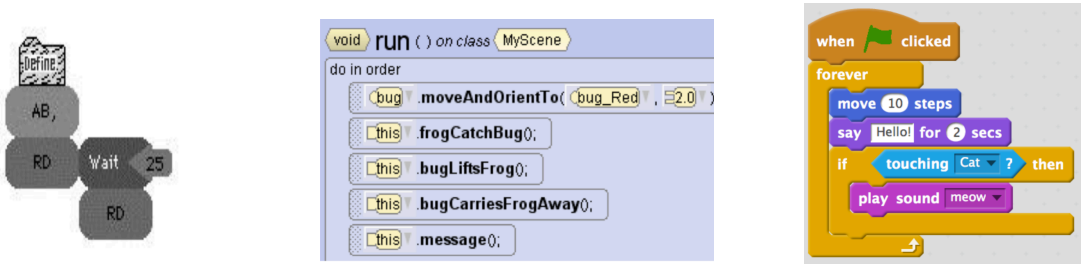
**Representation and Learning**

The tools we use have a profound, and often unforeseen, impact on how and what we think. 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. 17). The recognition that mental activity is mediated by tools and signs is one of the major contributions of the work of Vygotsky (1978, 1986) who argued that it is the external world that shapes internal cognitive functioning (Wertsch, 1991). In the context of programming languages, a number of studies have systematically explored the relationship between programming and other symbol systems (Sherin, 2001) as well as differences between programming languages (Gilmore & Green, 1984), delineating some of the ways in which the representational infrastructure shapes emerging understanding.

One oft overlooked aspect of representations and symbol system is the fact that they are not static but instead can change over time. Wilensky and Papert's (2010) theory of restructuration captures the shifting nature of representational systems and provides a set of criteria by which a given structuration can be evaluated. Given the rapidly shifting landscape of programming languages and representations, this framework provides a productive lens with which to investigate and evaluate programming tools.

**Block-based programming**

The block-based approach of visual programming (Figure 1) has become widespread in recent years with the emergence of a new generation of tools, led by the popularity of Scratch (Resnick et al., 2009), Alice (Cooper, Dann, & Pausch, 2000), and Blockly (Fraser, 2015). Block-based programming environments leverage a programming-primitive-as-puzzle-piece metaphor to provide visual cues to the user about how and where commands can be used as their means of constraining program composition. Programming in these environments takes the form of dragging blocks into a canvas and snapping them together to form scripts with numerous scaffolds provided to help the programmer (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010; Tempel, 2013; Weintrop & Wilensky, 2017). A growing body of empirical research is finding that these tools and the use of block-based programming is making programming and the powerful ideas of computing more accessible to and engaging for K-12 learners (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Franklin et al., 2017; Grover, Pea, & Cooper, 2015; Kelleher, Pausch, & Kiesler, 2007; Lewis, 2010; Maloney, Peppler, Kafai, Resnick, & Rusk, 2008; Price & Barnes, 2015; Weintrop & Wilensky, In Press, 2015).




LogoBlocks Alice Scratch

**Figure 1.** Three examples of block-based programming tools.

## Methods and Materials

In this section, we present details on the study, including the study design, programming environments used, and participants.

## Study Design and Data Collection Strategy

This study began on the first day of school and used a quasi-experimental setup with three high school introductory programming classes. Each of the three classes used a different Pencil.cc modality (block-based, text-based, or hybrid). This means students only saw one of the three modalities (so a student in the Blocks condition, never saw the Text or Hybrid interface). Over the course of the five-week study, four major conceptual topics were covered: variables, conditional logic, looping logic, and procedures.

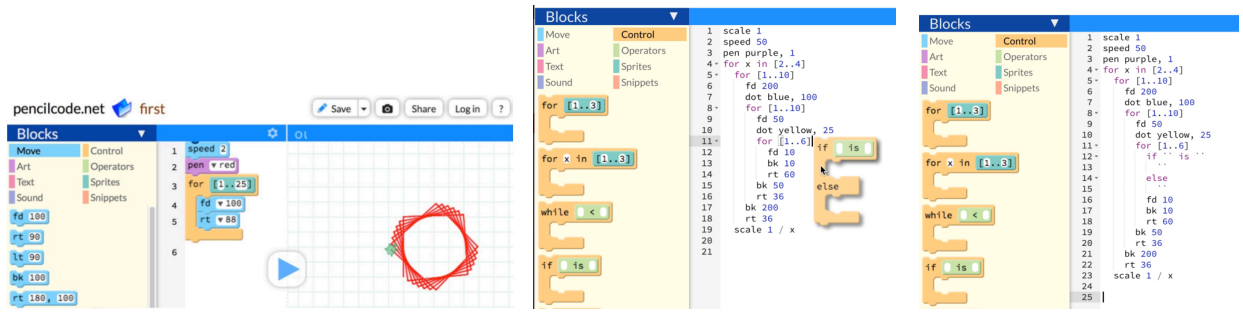
At the conclusion of the study, students were asked short answer questions about the four central programming concepts covered in the curriculum: variables, conditional logic, iterative logic, and functions. For each concept, students responded to the following open-ended prompt: "What do ___ do? And how are they used in programs?" Where the ___ in each question was replaced with "variables", "for loops and while loops", "if and if/else statements", and "functions". Student responses to these questions were open coded using a grounded theory approach (Strauss & Corbin, 1994), the results of which are presented in this paper.

## Setting and Participants

This study was conducted at a large, selective enrollment, urban, public high school in a Midwestern city. The computer science course used for the study is an elective class. A total of 90 students participated in the study. The self-reported racial breakdown of the participants was: 41% White, 27% Hispanic, 11% Asian, 11% Multiracial, and 10% Black. The three classes in the study were comprised of 15 female students and 75 male students. This gender disparity is problematic, but recruitment was out of the control of the researchers. Of the students participating in the study, 47% speak a language other than English in their households. A majority of the students in the school (58.6%) come from economically disadvantaged households. The experiment was conducted in an existing Introduction to Programming course with the same teacher taught all three sections of the course.

## Meet Pencil.cc

Pencil.cc is a customized version of the Pencil Code environment (Bau, Bau, Dawson, & Pickens, 2015) that supports block-based, text-based, and hybrid blocks/text programming, but is designed such that a given user only has access to one of these programming modalities. Pencil.cc's blocks interface (Figure 2a) features many of the defining features of block-based tools, including the drag-and-drop programming mechanism, a palette of blocks for the user to choose from, and visual cues on how and where blocks can be used. The text version of Pencil.cc presents users with a text editor that includes basic programming supports like highlighting, automatic formatting, and syntax checking. The hybrid form of Pencil.cc retains the block-palette and the ability to drag-and-drop commands into a program but replaces the blocks canvas with a text editor. When a user drags a block from the palette onto the text canvas, the blocks turn into the textual equivalent and is inserted into the program in a syntactically valid way. Thus, the hybrid interface supports both drag-and-drop and keyboard-driven composition. Figures 2b and 2c show Pencil.cc's hybrid interface. Aside from the programming modality, all other features of the three modes of Pencil.cc are the same, meaning the capabilities and expressive power of the environments are equivalent; anything that can be done in one environment can also be achieved in the other two.




(a) (b) (c)

**Figure 2.** Pencil.cc's interfaces. (a) shows the blocks interface while (b) and (c) show Pencil.cc's hybrid blocks/text interface. The middle image (b) shows how learners can drag-drop blocks into the text editor; the right image (b) shows the results of this action.

## Findings

This section presents the results of our analysis of students' responses to the "What do ____ do? And how are they used in programs?" questions that were posed at the end of the study. Due to length restrictions, we only present our analysis of the variables concept, the full manuscript will also include an analysis of conditional logic, iterative logic, and functions.

### Variables

The first question students responded to from this group was: "What do variables do? And how are they used in programs?" In analyzing the responses given, a variety of answers were given, more specifically, students employed a number of different metaphors in their descriptions of variables and their uses. In this analysis, we explore the different metaphors used to describe variables and discuss the implications of each.

The first type of response identified was to use the metaphor of a variable being like a container that stores things. For example, one students gave the response "*Variables are values that store information, they are used to store information in*[1]." The idea of a variable being a container that holds things is commonly used in computer science classrooms. The second metaphor found in student responses is similar to the container idea, but instead of holding the value, variables serve as placeholders for that value. In other words, the value is stored somewhere else and the variable serves as a representation used to get access to that value. This view can be seen in responses such as "*Variables are placeholders for something*" and "[Variables] *are something that stands in or represents something else*". A third metaphor that is similar to the first two is that of a variable as a pointer. While the idea of a variable as a pointer to a value could arguably be collapsed with the prior group because it shares the feel of a placeholder, it is left separate here because the notion of a pointer carries specific meaning in computer science. Fewer students described variables in this way relative to the first two categories (four in total), one of these responses reads: "*Variables are values that programs use to reference pieces of information in the code.*" The last type of response given by students was to not describe the variables in terms of the values they represent, but instead, to define the variable as an object in its own right, independent of what its value is. Sample responses from this category include: "V*ariables are things that change inside a function and are not things set in stone*" and "*Variables are anything you want them to be. They're used to tell the computer that another thing equals something else and so on*".

[1] Note: italicized text in this section denote direct quotes from the students' typed-in responses.

All student responses were coded as one of these four categories, with a few responses being coded as more than one when students drew on more than one metaphor within their response. Only five of the 82 responses did not fit into any of these four categories[2]. Figure 3 shows the distribution of student responses grouped by condition.

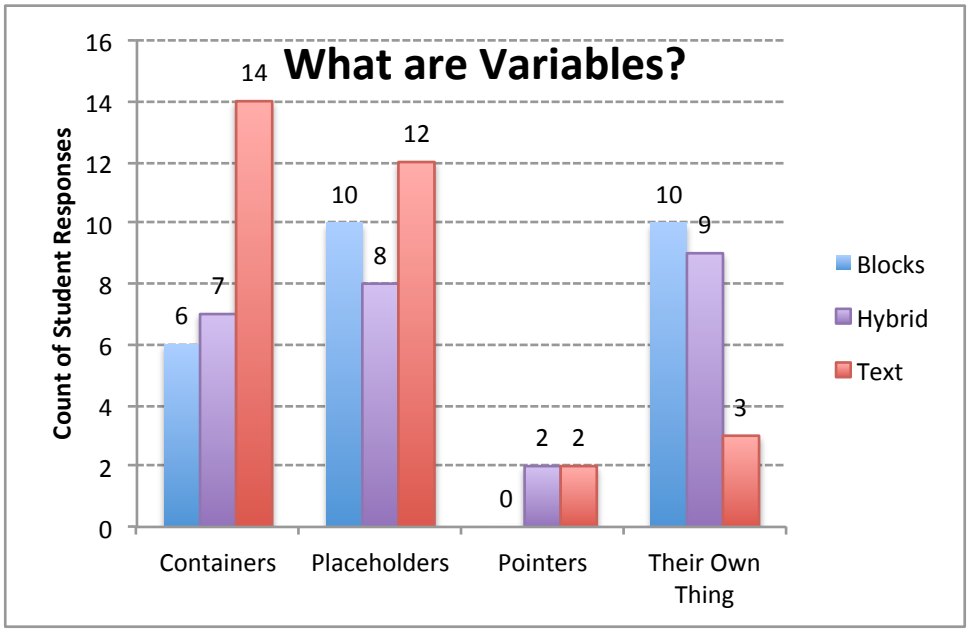



**Figure 3.** How students described variables, grouped by the form of Pencil.cc they used.

Learners in the Text condition preferred the variables-as-containers metaphor, followed closely by the placeholder metaphor, and were least likely to describe variables as pointers or as their own thing. Conversely, students in the Block condition were most likely to use the placeholder metaphor or see variables as their own thing and were much less likely to invoke the containers metaphor relative to their text-based peers. When looking at student responses to this conceptual question about the nature of variables, the Hybrid group aligns more closely with the Blocks condition, showing a higher frequency of treating variables as their own entity compared to text, and less likely to utilize the container metaphor favored by students in the Text condition.

**Variables Discussion**

One possible explanation for the higher frequency of the Blocks and Hybrid condition treating variables as their own distinct entity is in how they are presented to the user in the blocks palette. In block-based programming environments, variables are blocks that can be dragged into a program in the same way loops, conditionals, and, in the case of Pencil.cc, visual and movement commands. This presentation seems to lend itself to conceptually treating variables as objects in their own right. Alternatively, the container metaphor is less intuitive from the graphical layout of the commands given the fact that variables are not visually depicted as encapsulating, holding, or in any other way containing the value. Instead, in Pencil.cc's blocks interface, the variable identified lives on one side of an = with the value on the other. A possible explanation for the Text

[2] An example of a response that does not fit into any of the four metaphor categories is "*Variables are used to keep a clean and concise code in your program*", which does not attend to what a variable is, but instead describes how it is used.

condition's more frequent use of this explanation is that the variable-as-container metaphor is used explicitly in other courses taught by the teacher in these classrooms so, when students ask for help with variables, it seems plausible that her response would utilize this metaphor. The placeholder metaphor appears frequently across all three groups and is a perspective that aligns with how the reference materials embedded within Pencil.cc describes their use. It is important to note that none of these responses are necessarily incorrect, or more correct than the other, but instead, the differences are highlighted here to show how modality shapes emerging understandings of programming concepts.

## Conclusion

Block-based programming is increasingly the way that novices are being introduced to programming and the field of computer science. Given this increasing role, it is important we understand the conceptual implications of using this modality as part of a learners' first programming experience. This paper begins to explore this relationship with respect to central programming concepts. Specifically documenting different conceptual metaphors for understanding what variables are and how they are used in programs. In doing so, we contribute to our growing understanding of the implications of block-based programming and their strengths and drawbacks as a first introduction to the world of computer science.

## Reference


Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" Programming. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 25:1-15.

Bau, D., Bau, D. A., Dawson, M., & Pickens, C. S. (2015). Pencil Code: Block Code for a Text World. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 445–448). New York, NY, USA: ACM.

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.

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 *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 231–236). New York, NY, USA: ACM.

Fraser, N. (2015). Ten things we've learned from Blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)* (pp. 49–50).

Gilmore, D. J., & Green, T. R. G. (1984). Comprehension and recall of miniature programs. *International Journal of Man-Machine Studies*, *21*(1), 31–48.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237.

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

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals: Logo vs. Scratch. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 346–350). New York, NY.

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.

Price, T. W., & Barnes, T. (2015). Comparing Textual and Block Interfaces in a Novice Programming Environment (pp. 91–99). Presented at the ICER '15, ACM Press.

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.

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.

Strauss, A., & Corbin, J. (1994). Grounded Theory Methodology: An Overview. In *Strategies of Qualitative Inquiry* (pp. 158–183). Thousand Oaks, CA: Sage Publications, Inc.

Tempel, M. (2013). Blocks Programming. *CSTA Voice*, *9*(1).

Vygotsky, L. (1978). *Mind in society: The development of higher psychological processes*. (M. Cole, V. John-Steiner, S. Scribner, & E. Souberman, Eds.). Cambridge, MA: Harvard University Press.

Vygotsky, L. (1986). *Thought and language*. Cambridge, MA: MIT Press.

Weintrop, D., & Wilensky, U. (In Press). Comparing Blocks-based and Text-based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education (TOCE)*.

Weintrop, D., & Wilensky, U. (2015). Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 101–110). New York, NY, USA: ACM.

Weintrop, D., & Wilensky, U. (2017). How Block-based Languages Support Novices: A Framework for Categorizing Block-based Affordances. *Journal of Visual Languages and Sentient Systems*, *3*, 92–100.

Wertsch, J. V. (1991). *Voices of the mind: A sociocultural approach to mediated action*. Cambridge, MA: Harvard University Press.

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