Analogical Transfer Between Programming Languages: Scratch to Python

Anonymous CogSci submission

Abstract

Analogical transfer presents unique challenges in computer science education, particularly during the shift from visually scaffolded block-based languages to syntactically complex text-based languages. [structure mapping theory] This study is an exploratory look at analogical transfer between block-based (Scratch) and text-based (Python) programming languages among late elementary and middle-school students. 76 students proficient in Scratch with minimal Python experience completed an assessment measuring transfer of key programming constructs. Results indicated limited transfer following brief instruction emphasizing structural similarities. Students' perceived similarity of the languages predicted overall Python performance, suggesting students may recognize structural mappings between the two. However, in regression models, we did not find evidence that students transfer after controlling for general programming ability. Findings suggest that students need more targeted instruction to conceptualize structural relationships and support robust analogical transfer between block-based and text-based programming languages.

Keywords: add your choice of indexing terms or keywords; kindly use a semicolon; between each term

Introduction

Analogical transfer—the ability to apply knowledge from one context to another based on shared structural relationships—is fundamental to human learning (Gentner, 2003). Computer science (CS) education provides a unique opportunity to study this process, as students often are exposed to multiple programming languages throughout their education that differ in syntactic details but share underlying relationships that comprise the computational concepts (Kao et al., 2022). Increasingly, students are introduced to block-based languages that use visual interlocking blocks which represent code commands that snap together like puzzle pieces. While these block-based languages decrease barriers to entry by providing an intuitive, drag-and-drop interface for novice programmers, students are eventually expected to transition to text-based languages which can differ dramatically at the surface level of their syntactic details. Understanding how learners navigate surface differences to recognize deeper relational similarities between block-based and text-based languages remains a critical challenge for CS education research.

Transition from Block-based to Text-based Programming Languages

Block-based programming languages like Scratch have transformed introductory computer science by providing novices with visually intuitive, accessible entry points to coding (Kelleher & Pausch, 2005; Weintrop et al., 2018). For many students, these platforms represent their first encounter with programming and computational thinking (Bau et al., 2017; Resnick et al., 2009; Weintrop, 2019). However, as students progress, they are expected to transition to traditional text-based languages that lack the concrete scaffolding of block-based environments—a shift that presents significant learning challenges (Grover, 2021).

This transition is particularly difficult because it requires students to recognize conceptual equivalences across vastly different surface representations. Research on block-to-text transfer suggests some knowledge does carry over: programming patterns from Scratch often appear in students' Python code (Armoni et al., 2015). While some research (Gomez et al., 2019) has found that students with prior Scratch experience made fewer conceptual errors than students without such experience, other research (Armoni et al., 2015; Grover et al., 2015; Kazemitabaar et al., 2023; Weintrop et al., 2017) has found that students with prior Scratch experience bring misconceptions about loops and conditionals to the new language. Studies (Armoni et al., 2015; Gomez et al., 2019; Kazemitabaar et al., 2023) have also shown that syntax errors in the new language are a frequent challenge. The shift from color-coordinated blocks that visually define code structure to the nuanced syntactic requirements of text-based languages poses substantial obstacles for novice learners.

Theoretical Accounts of Programming Transfer

Structure-mapping theory (Gentner, 1983) proposes that analogical reasoning involves aligning the relational structure between two domains. The theory distinguishes between surface features—perceptual attributes like visual appearance, color, and specific labels—and relational features—abstract structural relationships and causal connections between elements. Successful analogical transfer depends on recognizing shared relational structure despite differences in surface features. However, humans often judge similarity based on surface matches rather than deeper relational commonalities, leading to transfer failure (Gick & Holyoak, 1983).

In G. Boleda, I. Dautriche, M. Maldonado, C. Saldana and T. Torrent (Eds.), *Proceedings of the 48th Annual Conference of the Cognitive Science Society*

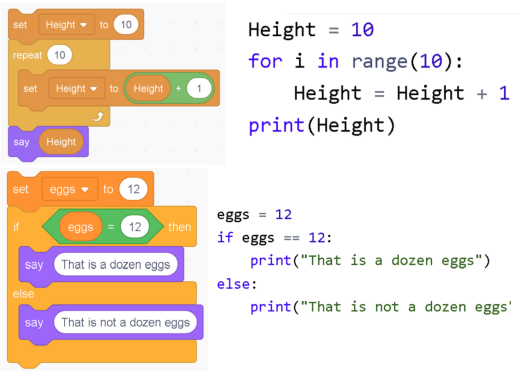

Figure 1: Analogous programming chunks in Scratch (left) and Python (right) illustrating the concepts of iteration (top) and conditionals (bottom).

Research on programming transfer suggests that, whereas experienced programmers are able to transfer at the conceptual level despite some syntactic differences (Wu & Anderson, 1990), novice programmers focus on syntactic details when learning new languages (Stefik & Siebert, 2013). This novice fixation on syntax over context mirrors the surface over structure bias often exhibited by novices in general research on transfer and problem solving (Chi et al., 1981). When code chunks differ in syntactic details they often result in spatial mismatches that are non-intuitive for alignment, such as Scratch's repeat block shown in Figure 1. Differences in spatial arrangements have been shown to greatly influence transfer in past research on analogy (Matlen et al., 2020; Simms et al., 2023; Zheng et al., 2022). Thus, spatial differences in the syntax and novice's surface-feature bias suggests a prediction: when text syntax resembles previously learned block syntax, novices will more readily transfer knowledge; when syntax differs substantially, transfer may fail even when the underlying concepts are identical.

The Present Investigation

In the present investigation, we aim to better understand the extent to which late elementary and middle-school students can transfer between Scratch and Python, two widely used languages in computing education. We focus on this age group as students at this age often switch from block-based to text-based languages. This work is part of a broader project that aims to design lessons to support students in transferring knowledge between these languages.

To these ends, we developed an assessment to measure transfer with minimal explicit instruction. The assessment targets students proficient in Scratch but with little to no Python experience. It provides brief instruction highlighting syntactic similarities and differences between Scratch and Python for core programming concepts (such as variables, conditionals, and iteration), then measures students' ability to solve related problems in Python (McKee et al., 2025).

Our research questions were the following:

1. With minimal instruction on the similarities and differences between Scratch and Python, to what extent can students transfer their Scratch knowledge to solve problems with analogous code in Python?
2. Does the surface similarity of the syntax predict transfer from Scratch to Python?

This assessment serves a practical aim in that it indicates for which concepts instruction should most heavily target for transfer support. For example, if students transfer readily for some concepts, it suggests that less instructional time is needed to support transfer. If students struggle to transfer for certain concepts, this would indicate where to target instructional support.

Because our primary aim was to understand transfer between these naturalistic languages we were not able to systematically manipulate similarity. However, to explore whether surface similarity was associated with transfer, we implemented two strategies: 1) we explicitly asked students to rate the similarity between Scratch and Python concepts, allowing us to explore associations between transfer performance are related to perceived similarity, and 2) to formalize our a priori predictions we developed a coding scheme that attempted to quantify the surface differences between the code for each concept. For each of the survey's six code chunk pairs used to illustrate Scratch and Python differences, we mapped symbols between the languages based on the relational structure and noted whether there were syntactic differences. Syntactic differences could be instantiated either by mismatches in the spatial arrangement of analogous symbols (e.g., in Figure 1, whereas set and to symbols in Scratch correspond with the = symbol in Python, set appears to the left of the variable in Scratch), or by symbols that do not share a match with the other language (e.g., the absence of a repeat arrow in Python in Figure 1). Research on analogy also suggests that cross mappings—when the same objects play different relational roles across relational systems—are particularly challenging (e.g., Markman and Gentner, 1993). Thus, we noted whether each code chunk contained a cross mapping, a common symbol that serves a different function across the languages (e.g., the single = symbol in the iteration examples in Figure 1). Two members of the research team independently coded the code chunks and came to 86% intercoder agreement with differences resolved through discussion. This generated the codes presented in Table 1.

Table 1: Similarity by Concept.

Concept	Syntactic Differences	Cross Mappings
Variable	1	0
Numeric	1	0
Boolean	1	1
Iteration	2	0
Character	2	0
Conditional	2	1

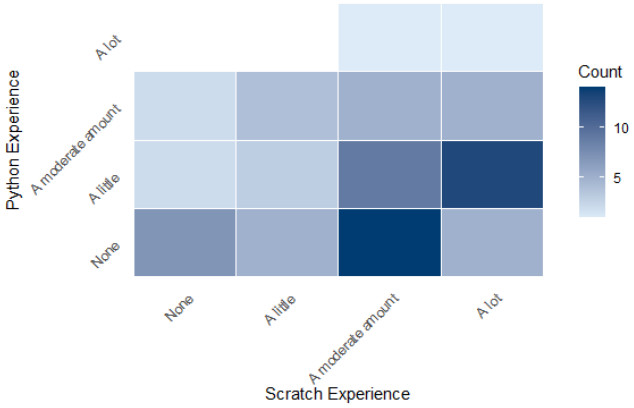

Figure 2: Self-reported Scratch and Python Experience.

Based on this analysis, we expected that students would be more likely to transfer for the Variable and Numeric concepts, and less likely to transfer for the Boolean, Iteration, Strings, and Conditional concepts.

Methods

Participants

Participants were recruited from a California-based nonprofit that provides coding instruction to students aged 5–16 through afterschool programs and camps. The assessment was administered to late elementary and middle-school afterschool programs during the 2024-2025 and 2025-2026 school years. A total of 86 students participated in the assessment. 10 students were removed from the sample for not finishing the assessment. Students had a mean age of 10.81 years (SD = 1.42) and 56.58% of students were male.

Assessment

The assessment includes three sections: demographics, self-reported prior programming experience (see Figure 2), and six sets of transfer questions on fundamental CS concepts. The transfer questions consist of 1) a Scratch pre-test item, 2) a section showing the alignment of Scratch code and equivalent Python code (see Figure 3), 3) a question to rate how similar participants think the Scratch and Python code are (Likert scale: 0 = "Very Different", 3 = "Very Similar"), and finally 4) three Python questions covering conceptual, semantic, and syntactic transfer. The assessment was administered by the instructors of the programs via Qualtrics during a class period. A researcher was present to oversee the student assent process and answer student questions.

Results

Psychometric results

Reliability analyses were conducted to evaluate the internal consistency of the assessment using Cronbach's alpha. A coefficient of .65 or greater was considered acceptable. The

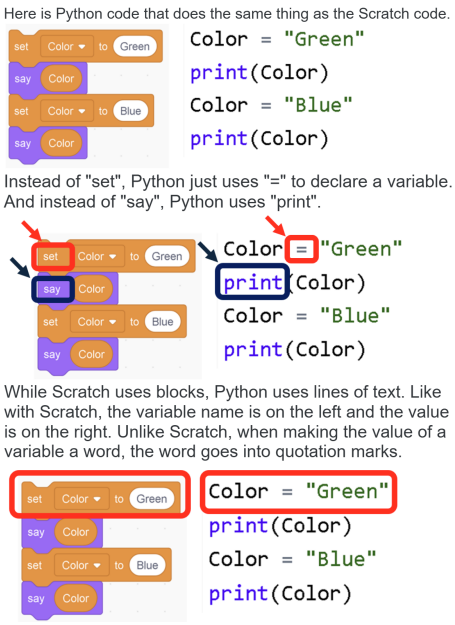

Figure 3: Alignment instructions for variable declaration. First students are presented with equivalent Scratch and Python code (top), then lexical comparison is presented for that code (middle), and finally the syntax is compared (bottom).

complete transfer assessment demonstrated high internal consistency, with Cronbach's alpha of α = .84, suggesting reliable measurement of programming knowledge. Item-total correlations for all items exceeded the acceptable threshold, and removal of any single item did not result in improved reliability.

Subscale analyses indicated that the Python subscale (18 items) demonstrated good reliability (α = .80). One item assessing syntactic transfer for iteration was negatively correlated with the remaining items (item-total correlation r = -.04) and was commonly answered incorrectly. Excluding this item slightly increased the subscale reliability to α = .81.

The Scratch subscale (6 items), however, exhibited lower internal consistency (α = .54). Removing one item related to Boolean logic, which showed poor item-total correlation (r = .01), increased the subscale's reliability to α = .62. Although this subscale has a relatively low alpha, this is at least partially attributable to the limited number of items. The average inter-item correlation of the 5-item subscale is .25, which is generally considered acceptable.

These findings indicate that the assessment provides robust measurement of students' conceptual understanding of Scratch and Python for exploratory research, though the Scratch subscale may benefit from further refinement to improve reliability.

Assessment results

Similarity Scores A multiple linear regression was conducted to examine the predictors of total Python scores based on total similarity ratings and levels of previous Python experience. If similarity scores are positively associated with overall Python performance, students who recognize the similarities between languages transfer more. The model was significant, $F(4, 71) = 5.45$, $p < .001$, $R^2 = .23$. Of the individual predictors, total similarity rating ($t = 3.39$, $p = .001$) and students reporting "a little" experience in Python compared to no experience ($t = 2.60$, $p = .011$) were significant, but neither "a moderate amount" (t=0.86, $p = .39$) nor "a lot" ($t = -0.08$, $p = .94$) of Python experience were statistically significant predictors of Python scores. Similarity ratings and limited prior Python experience are significant contributors to Python scores, while more extensive experience does not show a statistically significant impact. This suggests that students who see more similarity between the languages transfer more overall.

Transfer by Programming Construct Multiple regression analyses were used for each construct to test if students' score on the Scratch item predicted their performance on the Python items for that construct. If students are able to transfer their knowledge to the new language, we would expect to see a positive relationship between construct-specific Scratch scores and Python scores. For all constructs except for Boolean logic, Scratch scores were a positive and statistically significant predictor of Python performance (see Table 2).

Total Scratch score was then added to the model as a predictor to control for general programming ability. Total Scratch score was statistically significant for all constructs except for variable declaration (Table 3). In this model, construct-specific Scratch score was not statistically significant. These results suggest that general programming ability was a stronger predictor than scratch score on the concept of Python performance.

Finally, students' similarity rating for the construct was added to the model, as an interaction with total Scratch score (see Table 4). If the interaction between these two predictors was positively related with Python performance, this would suggest that students who answer the Scratch pre-test item correctly perceive greater similarity between the code segments, an indication they may be solving python questions based on relational correspondences. This interaction, however, was only statistically significant and positive for variable declaration, but not for the remaining constructs.

For all constructs, a higher proportion of students who answered the Scratch pre-test item correctly achieved full transfer (all three items answered correctly) or partial transfer (two items answered correctly) to Python than of those who did not (see Figure 4 and Figure 5).

Of the constructs with higher surface similarity (i.e., variable declaration and numeric operations), a subset of students who answered the Scratch pre-test item correctly achieved full

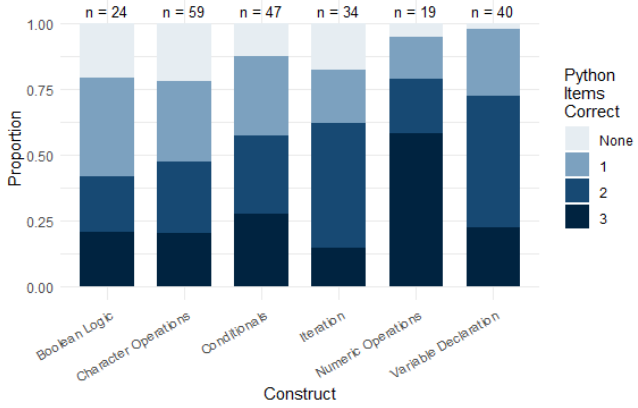

Figure 4: Python Scores by Construct (Scratch Correct)

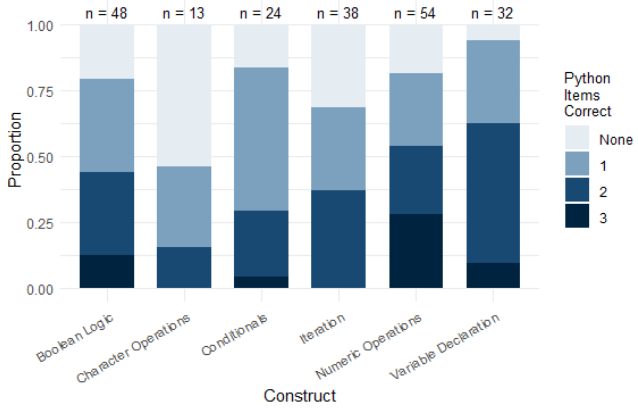

Figure 5: Python Scores by Construct (Scratch Incorrect)

	Variables	Numeric	Characters	Booleans	Conditionals	Iteration
Construct Scratch Score	0.27	0.69 *	0.84 **	0.06	0.56 **	0.54 *
	(0.18)	(0.28)	(0.31)	(0.25)	(0.24)	(0.21)
Constant	1.66 ***	1.63 **	0.62 *	1.35 ***	1.17 ***	1.05 ***
	(0.13)	(0.14)	(0.28)	(0.14)	(0.19)	(0.15)

Table 2: Results of a linear regression model testing the relationship between construct-specific Scratch scores and Python performance. (*** p < 0.001; ** p < 0.01; * p < 0.05)

	Variables	Numeric	Characters	Booleans	Conditionals	Iteration
Total Scratch Score	0.11	0.36 ***	0.28 **	0.32 ***	0.31 ***	0.29 **
	(0.06)	(0.09)	(0.09)	(0.07)	(0.09)	(0.09)
Construct Scratch Score	0.19	0.27	0.47	0.04	0.27	0.23
	(0.18)	(0.27)	(0.32)	(0.22)	(0.23)	(0.22)
Constant	1.42 ***	0.73 **	0.30	0.49 *	0.61*	0.45
	(0.19)	(0.26)	(0.29)	(0.23)	(0.24)	(0.23)

Table 3: Results of a multiple regression model testing the relationship between total Scratch scores, construct-specific Scratch scores, and Python performance. (*** p < 0.001; ** p < 0.01; * p < 0.05)

	Variables	Numeric	Characters	Booleans	Conditionals	Iteration
Total Scratch Score	0.11	0.36 ***	0.30 **	0.32 ***	0.27 **	0.31 **
	(0.07)	(0.09)	(0.09)	(0.07)	(0.09)	(0.09)
Construct Scratch Score	-0.86	0.44	0.91	-0.04	-0.20	0.36
	(0.57)	(0.91)	(0.67)	(0.65)	(0.65)	(0.55)
Construct Scratch Score:Similarity Rating	0.54 *	-0.11	-0.33	0.09	0.20	-0.03
	(0.27)	(0.41)	(0.36)	(0.33)	(0.32)	(0.28)
Constant	1.92 ***	0.47 **	-0.50	0.55	0.57	0.68
	(0.43)	(0.42)	(0.59)	(0.52)	(0.48)	(0.25)

Table 4: Results of a multiple regression model testing the relationship between total Scratch scores, construct-specific Scratch scores, their interaction with similarity ratings, and Python performance. (*** p < 0.001; ** p < 0.01; * p < 0.05)

or partial transfer to Python. For variable declaration, 55.6% of students answered the Scratch item correctly, 22.5% of whom achieved full transfer compared to 9.4% of those who did not pass the Scratch pre-test answering all Python questions correctly. For numeric operations, while only 26% of students answered the Scratch item correctly, more than half (57.9%) of this group achieved full transfer to Python, compared to 4.2% of those who did not pass the Scratch pre-test. Most students had the same misconception on the Scratch pre-test: 54.7% selected an answer where the variable stored an expression (e.g., "2 + 3") rather than a calculated value (e.g., "5").

Of the constructs with lower surface similarity, 20.3% of students who answered the Scratch pre-test correctly achieved full transfer for character operations and 27.7% for conditionals, compared to none and 4.2%, respectively, from those who did not pass the Scratch pre-test. For Boolean logic, students who answered the Scratch pre-test accurately showed similar proportions of failed, partial, and full transfer as those who did not. The same misconception from numeric operations was seen with this construct: more than half of students (54.2%) chose unevaluated expressions in Scratch, and a similar proportion (48.6%) made this error in Python.

For iteration, 14.7% of students achieved full transfer of those who accurately answered the Scratch item, while no students in the comparison group did. Many students (41.7%) also selected the unevaluated expression in this Scratch pre-test. In Python, common errors included omitting iteration altogether (32.4%) or printing only the final variable value (30.9%). Most (69%) students answered the conceptual question correctly, tracing the loop successfully when no variable updating was required.

Discussion

With minimal instruction on the similarities and differences between Scratch and Python, to what extent can students transfer their Scratch knowledge to solve problems with analogous code in Python?

Transfer with minimal instruction was not robust. Though scratch score on the tested construct predicted scores on that construct, Scratch score as an indicator of general programming ability was the best predictor of construct-specific Python performance. These results suggest that general programming ability determined Python outcomes and that students were not consistently transferring construct-specific knowledge from Scratch.

These results align with previous studies showing transfer difficulties between block-based languages when learning a text-based language (Armoni et al., 2015; Grover et al., 2015; Kazemitabaar et al., 2023; Weintrop et al., 2017). Even with explicit instruction on the similarities and differences between the concepts, students struggled to transfer knowledge robustly. This indicates that students need greater scaffolding, likely utilizing techniques from the analogical literature (e.g., Vendetti et al., 2015) to benefit from their prior knowledge when learning related concepts in a new language.

Does the surface similarity of the syntax predict transfer from Scratch to Python?

Total similarity ratings had a significant positive relationship to Python scores, suggesting that students who see more similarity between the languages also perceive the structural similarities.

However, given that construct-specific Scratch knowledge was not a predictor of Python performance, our results do not show that students transferred more for constructs with high surface similarity in comparison to those with low surface similarity. However, comparisons involved between a single pair of languages with a low amount of variation, thus, the lack of transfer by similarity does not rule out the possibility that transfer might differ when comparing between multiple language pairs that differ in similarity. The interaction of similarity ratings and construct-specific Scratch score was a predictor of transfer for variable declaration, one construct with high surface similarity. For the other constructs, surface similarity ratings were not found to be a significant predictor of Python performance. Students need more instruction to develop the structural mappings necessary to transfer reliably, even for constructs with high surface similarity.

We are in the process of conducting other studies to further make sense of these findings, including cognitive interviews with the same items. This data will allow us to take a deeper look at how students perceive similarity between problems during transfer opportunities.

Limitations

Given that this study investigated transfer between naturalistic programming languages, traditional experimental manipulation was not possible. However, qualitative analysis of student errors shows that some Scratch pre-test items included common misconceptions when the corresponding Python items did not, for instance, with the numeric operations construct items. Previous research (Kohn, 2017; Žanko et al., 2023) found that students commonly believe that variables store the expression rather than the calculated value, such as was the case in our data. This misconception was also present in the Boolean logic items, where about half of students selected the unevaluated expression in both Scratch and Python. Thus, future research could include assessment items that include common coding misconceptions in Scratch pre-test and Python transfer questions for each construct.

Acknowledgments

This work is supported by an anonymous funder. Additional funder disclaimer text and acknowledgements will go here.

References

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)*, *14*(4), 1–15.

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: Blocks and beyond. *Communications of the ACM*, *60*(6), 72–80.

Chi, M. T., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive science*, *5*(2), 121–152.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive science*, *7*(2), 155–170.

Gentner, D. (2003). Why we're so smart. *Language in mind: Advances in the study of language and thought*, *195235*.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive psychology*, *15*(1), 1–38.

Gomez, M. J., Moresi, M., & Benotti, L. (2019). Text-based programming in elementary school: A comparative study of programming abilities in children with and without block-based experience. *Proceedings of the 2019 ACM Conference on Innovation and technology in computer science education*, 402–408.

Grover, S. (2021). Teaching and assessing for transfer from block-to-text programming in middle school computer science. In *Transfer of learning: Progressive perspectives for mathematics education and related fields* (pp. 251–276). Springer.

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.

Kao, Y., Matlen, B., & Weintrop, D. (2022). From one language to the next: Applications of analogical transfer for programming education. *ACM Transactions on Computing Education*, *22*(4), 1–21.

Kazemitabaar, M., Chyhir, V., Weintrop, D., & Grossman, T. (2023). Scaffolding progress: How structured editors shape novice errors when transitioning from blocks to text. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 556–562.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM computing surveys (CSUR)*, *37*(2), 83–137.

Kohn, T. (2017). Variable evaluation: An exploration of novice programmers' understanding and common misconceptions. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 345–350.

Markman, A. B., & Gentner, D. (1993). Structural alignment during similarity comparisons. *Cognitive psychology*, *25*(4), 431–467.

Matlen, B. J., Gentner, D., & Franconeri, S. L. (2020). Spatial alignment facilitates visual comparison. *Journal of Experimental Psychology: Human Perception and Performance*, *46*(5), 443.

McKee, K., Matlen, B., Owen, R., Caballero, E., Houchins, J., & Kao, Y. (2025). Capturing student's spontaneous knowledge transfer between block and text-based programming languages. *Proceedings of the Annual Meeting of the Cognitive Science Society*, *47*.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67.

Simms, N. K., Matlen, B. J., Jee, B. D., & Gentner, D. (2023). Spatial alignment supports comparison of life science images. *Journal of Experimental Psychology: Applied*, *29*(4), 747.

Stefik, A., & Siebert, S. (2013). An empirical investigation into programming language syntax. *ACM Transactions on Computing Education (TOCE)*, *13*(4), 1–40.

Vendetti, M. S., Matlen, B. J., Richland, L. E., & Bunge, S. A. (2015). Analogical reasoning in the classroom: Insights from cognitive science. *Mind, Brain, and Education*, *9*(2), 100–106.

Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, *62*(8), 22–25.

Weintrop, D., Bain, C., Wilensky, U., & Education, U. (2017). Blocking progress? transitioning from block-based to text-based programming. *Proceedings of the American Educational Research Association*, 1–8.

Weintrop, D., Killen, H., & Franke, B. E. (2018). Blocks or text? how programming language modality makes a difference in assessing underrepresented populations. International Society of the Learning Sciences, Inc.[ISLS].

Wu, Q., & Anderson, J. R. (1990). *Problem-solving transfer among programming languages* (tech. rep.).

Žanko, Ž., Mladenović, M., & Krpan, D. (2023). Mediated transfer: Impact on programming misconceptions. *Journal of computers in education*, *10*(1), 1–26.

Zheng, Y., Matlen, B., & Gentner, D. (2022). Spatial alignment facilitates visual comparison in children. *Cognitive Science*, *46*(8), e13182.