

AN EMPIRICAL INVESTIGATION INTO THE TRANSITIONAL FRICTION OF BLOCK-BASED  
PROGRAMMING LANGUAGES

By

Alex Hoffman

Bachelor of Science in Computer Science  
Hardin Simmons University  
2000

Master of Business Administration  
University of Nevada, Las Vegas  
2018

A dissertation submitted in partial fulfillment of  
the requirements for the

Doctor of Philosophy – Computer Science

Department of Computer Science  
Howard R. Hughes College of Engineering  
The Graduate College

University of Nevada, Las Vegas  
May 2024

© Alex Hoffman, 2024

All Rights Reserved



## **Dissertation Approval**

The Graduate College  
The University of Nevada, Las Vegas

April 11, 2024

This dissertation prepared by

Alex Hoffman

entitled

An Empirical Investigation into the Transitional Friction of Block-Based Programming Languages

is approved in partial fulfillment of the requirements for the degree of

Doctor of Philosophy – Computer Science  
Department of Computer Science

Andreas Stefik, Ph.D.  
*Examination Committee Chair*

Hal Berghel, Ph.D.  
*Examination Committee Member*

Laxmi Gewali, Ph.D.  
*Examination Committee Member*

Fatma Nasoz, Ph.D.  
*Examination Committee Member*

Gregory Moody, Ph.D.  
*Graduate College Faculty Representative*

Alyssa Crittenden, Ph.D.  
*Vice Provost for Graduate Education &  
Dean of the Graduate College*

# Abstract

Computers are ubiquitous in our lives, so skilled workers are needed to create the software these devices require to operate. Computer science education, and more broadly, computational thinking are critical to sustaining innovation and economic growth. Much of the research on early computer science education focuses on block-based programming languages, a subset of visual programming languages. The intent behind block languages is to provide novices an introduction into computer science principles and programming in general, but there is a longstanding issue with learners' transitional friction from block-based languages to the more professionally used text-based languages. This makes the current use of block languages predominantly narrow in scope: teaching just the basics of programming. Barring the rare exceptions of using block languages for a full course, most pedagogy utilizes block-based programming for a matter of mere weeks before transitioning to text-based languages. Students initially learning with block modalities typically learn programming fundamentals faster than those who start with text modalities, but the differences tend to level off after the block-based learners transition to text-based programs, which calls into question the benefits of starting in blocks. This transitional friction from block languages to text languages arguably hinders the educational scaffolding required to develop students into professionals, so addressing the sources of this friction is paramount to the educational process.

The first original empirical study is an investigation into the interaction of visual attributes of block-based programming languages, which could help or hinder a programmer's ability to read the program's code. The techniques were applied in a study of 30 professional programmers and 57 students from the University of Nevada, Las Vegas for a total of 87 participants across four treatment groups. In that first

empirical study, we were able to quantitatively determine that colors and color categorization are beneficial for both novices and professionals. We recommend to have block-based programming environments employ a left-margin aligned programming interface with even spacing instead of an open palette. Distinct block shapes appear to be beneficial, although more testing is needed to determine what shapes make most sense. Adding the capability to have comments and line numbers would be beneficial for novices and professionals alike. This analysis lends itself to ongoing development on existing block languages to ease current learners' transition from block languages to text languages.

The second original empirical study is an investigation into the features and functionality, or tools, of an integrated development environment (IDE) for a block-based language. The study particularly focuses on what an IDE needs to have available for developers to be able to perform adequately as well as what types of context-aware capabilities an IDE could have to accelerate development practices. The study consisted of 21 professional programmers and 102 students from the University of Nevada, Las Vegas for a total of 123 participants, all of whom were in a single group, and we analyzed their responses both together and per experience group. The results indicate that a context-aware IDE for a block-based language would be beneficial; thus, we recommend different sets of tools depending on the programming scenario. One important finding is that, regardless of scenario, developers consistently rated example code as one of the most helpful tools. We began the investigation hypothesizing that novices and professionals might have very different needs. After our survey, however, we noted far more overlap than we expected in terms of the type of information that both groups claimed were helpful. The key contribution of this research project is to investigate the initial key components of a block-based language and accompanying IDE that could scale in utilization from very young learners up through to professional programming activities. If such a language and environment existed, it might reduce the transitional friction between products that the community thought were only for children to those we presume might be only for professionals.

# Acknowledgements

First, I thank my committee for their guidance and effort in helping me achieve this goal. I express my deepest appreciation and gratitude for my advisor, Dr. Andreas Stefik. Many times throughout this process, I felt like you were my only hope to complete this journey. Anytime I felt that it was impossible, your support helped me stay on target. Your work inspires me to be better, both as a researcher and a human. I look forward to continuing this research journey together. I thank Dr. Hal Berghel, who provided me the initial spark of excitement for research even before I started this journey and for providing a shining example of a life well-lived in service of educating others. I thank Dr. Laxmi Gewali for his all his wisdom, encouragement, and everything he does for the CS department at UNLV. I send a special thanks to Dr. Fatma Nasoz who advised and supported me throughout the years despite the pandemic and circumstance preventing us from working directly together; hopefully we can collaborate in my next phase. Finally, I express my sincerest thanks and appreciation to Dr. Greg Moody, who has advised, supported, and educated me since my masters degree. I appreciate you being there for me all of these years, and I hope to continue working together throughout our careers.

In addition, I thank Dr. Sidkazem Taghva for the vote of confidence in allowing me to teach classes in the CS department for so many years. To Dr. Nadine Bentis and Leith Martin, I extend my deepest appreciation for all the support you provided me in this journey; I am eternally grateful. Thank you to the rest of the UNLV CS, MIS, MBA, and Finance professors from whom I learned so much, and thank you to Dr. Paul LaPlante and Dr. Benjamin Cisneros who supported my research.

Thank you to all my friends and peers who helped me achieve this goal. Starr, life is weird; thank you

for supporting me even though it was no longer an expectation, which in my opinion just makes it all the more real. Thank you for your heartfelt counsel, reaching out to help, and giving good advice. Hannah, thank you for your insight, feedback, and encouragement. You have the makings of being a world-changing researcher; I look forward to working with you for many years to come. To my lab-mates, fellow PhD students, co-instructors, and peers who toiled with me in this journey, thank you for the commiseration and camaraderie. I learned and grew from each of you. Jared, James, Alexey, Bryce, Mana, Tihleigh, John, Mariela, and of all my other friends around the globe, your words of encouragement and support have kept me going.

*“Your focus determines your reality”* – Qui-Gon Jinn

*“So long, and thanks for all the fish”* – Douglas Adams

ALEX HOFFMAN

*University of Nevada, Las Vegas*

*May 2024*

# Table of Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>Table of Contents</b>	<b>vii</b>
<b>List of Tables</b>	<b>xi</b>
<b>List of Figures</b>	<b>xii</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Research Questions . . . . .	6
1.3 Contributions . . . . .	6
1.3.1 Summary of Results . . . . .	7
<b>Chapter 2 Review of Literature</b>	<b>9</b>
2.1 Overview . . . . .	9
2.2 Literature Distribution . . . . .	9
2.3 Visual Programming Languages . . . . .	10
2.4 What Are Block-Based Programming Languages? . . . . .	12
2.5 Why Block Languages? . . . . .	14
2.6 Challenges with Blocks . . . . .	16
2.6.1 Self-Efficacy & Learning . . . . .	16
2.6.2 Accessibility . . . . .	17
2.7 Transitioning from Blocks to Text . . . . .	17



2.8	Computational Thinking . . . . .	19
2.9	Blocks for Educating Educators . . . . .	20
2.10	Blocks Beyond Formal Education . . . . .	20
2.11	Professional Software Engineers . . . . .	21
2.12	The State of Block Editors and IDEs . . . . .	22

**Chapter 3 Study 1: Assessing Visual Attributes’ Role in Readability and Comprehension of Block-Based Programs 24**

3.1	Methods . . . . .	24
3.1.1	Hypotheses . . . . .	24
3.1.2	Inclusion and Exclusion . . . . .	25
3.1.3	Participant Characteristics . . . . .	25
3.1.4	Sampling Procedures . . . . .	25
3.1.5	Sample Size, Power, and Precision . . . . .	25
3.1.6	Measures and Covariates . . . . .	26
3.1.7	Data Collection . . . . .	27
3.1.8	Quality of Measurements . . . . .	27
3.1.9	Instrumentation . . . . .	28
3.1.10	Masking . . . . .	29
3.1.11	Psychometrics . . . . .	30
3.1.12	Random Assignment Method . . . . .	30
3.1.13	Random Assignment Implementation and Concealment . . . . .	30
3.1.14	Data Diagnostics . . . . .	30
3.1.15	Analytic Strategy . . . . .	31
3.2	Results . . . . .	31
3.2.1	Participant Flow . . . . .	31
3.2.2	Recruitment . . . . .	31
3.2.3	Statistics and Data Analysis . . . . .	32
3.2.4	Timing and Correctness . . . . .	42
3.3	Discussion . . . . .	45
3.3.1	Support of Original Hypotheses . . . . .	45

3.3.2	Hypothesis 1 . . . . .	45
3.3.3	Hypothesis 2 . . . . .	47
3.3.4	Hypothesis 3 . . . . .	47
3.3.5	Hypothesis 4 . . . . .	48
3.3.6	Similarity of Results . . . . .	48
3.3.7	Interpretation . . . . .	48
3.3.8	Generalizability . . . . .	49
3.3.9	Implications . . . . .	50
<b>Chapter 4 Study 2: Investigating a Context-Aware Palette for a Block-Based Language</b>		<b>51</b>
4.1	Background . . . . .	51
4.2	Methods . . . . .	52
4.2.1	Hypotheses . . . . .	52
4.2.2	Inclusion and Exclusion . . . . .	52
4.2.3	Participant Characteristics . . . . .	53
4.2.4	Sampling Procedures . . . . .	53
4.2.5	Sample Size, Power, and Precision . . . . .	54
4.2.6	Measures and Covariates . . . . .	54
4.2.7	Data Collection . . . . .	55
4.2.8	Quality of Measurements . . . . .	55
4.2.9	Instrumentation . . . . .	56
4.2.10	Masking . . . . .	57
4.2.11	Psychometrics . . . . .	57
4.2.12	Participant Selection . . . . .	58
4.2.13	Data Diagnostics . . . . .	58
4.2.14	Analytic Strategy . . . . .	58
4.3	Results . . . . .	59
4.3.1	Participant Flow . . . . .	59
4.3.2	Recruitment . . . . .	61
4.3.3	Statistics and Data Analysis . . . . .	61
4.4	Discussion . . . . .	77

4.4.1	Support of Original Hypotheses . . . . .	77
4.4.2	Hypothesis 1 . . . . .	77
4.4.3	Hypothesis 2 . . . . .	79
4.4.4	Hypothesis 3 . . . . .	80
4.4.5	Similarity of Results . . . . .	81
4.4.6	Interpretation . . . . .	81
4.4.7	Generalizability . . . . .	92
4.4.8	Implications . . . . .	92
4.4.9	Limitations . . . . .	93
<b>Chapter 5</b>	<b>Conclusion</b>	<b>94</b>
5.1	Discussion of Research Questions . . . . .	94
5.1.1	RQ 1 . . . . .	94
5.1.2	RQ 2 . . . . .	95
5.1.3	RQ 3 . . . . .	96
5.2	Future Research . . . . .	99
<b>Appendix A</b>	<b>Survey Instrument 1</b>	<b>101</b>
<b>Appendix B</b>	<b>Survey Instrument 2</b>	<b>199</b>
	<b>Bibliography</b>	<b>214</b>
	<b>Curriculum Vitae</b>	<b>228</b>

# List of Tables

3.1	Groups . . . . .	26
3.2	Attributes Assessed . . . . .	27
3.3	Group Attributes . . . . .	29
3.4	Participants . . . . .	32
3.5	Task Correctness Between Professionals and Novices . . . . .	45
4.1	Participants . . . . .	53
4.2	IDE Potential Types of Tools . . . . .	55
4.3	Question 1: Use Statement Means/SD . . . . .	62
4.4	Question 2: Syntax Error Means/SD . . . . .	65
4.5	Question 3: If Statement Means/SD . . . . .	68
4.6	Question 4: Add Function Means/SD . . . . .	71
4.7	Question 5: Create Dataframe Means/SD . . . . .	74
4.8	Question 6: Chart Display Means/SD . . . . .	76
4.9	Count of Times Each Tool Ranks in the Top or Bottom Tools Across All Scenarios . . . . .	81
4.10	Sum of the Means . . . . .	82
4.11	Tool Median Weight Greater Than Zero . . . . .	82

# List of Figures

2.1	Research By Group . . . . .	10
2.2	Glinert’s Blox, an Early Visual Programming Language [1] . . . . .	11
2.3	Scratch, a Block-Based Programming Language Designed for Children [2] . . . . .	12
2.4	Blockly, Google’s Block-Based Programming Language [3] . . . . .	13
3.1	Colorful, In-line, Vertical Scroll . . . . .	28
3.2	Grayscale, Dispersed, Horizontal Scroll . . . . .	29
3.3	Block Color: Color Attribute Rating by Group and Experience . . . . .	33
3.4	Shape of the Blocks: Block Shape Attribute Rating by Group and Experience . . . . .	35
3.5	Arrangement of the Blocks: Block Arrangement Attribute Rating by Group and Experience . . . . .	37
3.6	Scrolling Direction: Scroll Direction Attribute Rating by Group and Experience . . . . .	38
3.7	Text, Punctuation, & Symbols: Text, Punctuation, & Symbols Attribute Rating by Group and Experience . . . . .	39
3.8	Lack of Comments: Lack of Comments Attribute Rating by Group and Experience . . . . .	40
3.9	Lack of Line Numbers: Lack of Line Numbers Attribute Rating by Group and Experience . . . . .	41
3.10	Spacing: Block Spacing Attribute Rating by Group and Experience . . . . .	43
3.11	Dropdown Arrow: Dropdown Arrow Attribute Rating by Group and Experience . . . . .	44
3.12	Task Timing by Group: Duration per Task by Group and Experience . . . . .	46
4.1	Categorization and Rank Questions . . . . .	56
4.2	Tufte Box Plot Example . . . . .	60
4.3	Question 1: Which Tools Would Be Helpful? . . . . .	62
4.4	Question 1: Code Snippet Displayed to Participant . . . . .	64
4.5	Question 1: Potential Partial Palette Based on Weights . . . . .	64

4.6	Question 2: Which Tools Would Be Helpful? . . . . .	65
4.7	Question 2: Code Snippet Displayed to Participant . . . . .	67
4.8	Question 2: Potential Partial Palette Based on Weights . . . . .	67
4.9	Question 3: Which Tools Would Be Helpful? . . . . .	68
4.10	Question 3: Code Snippet Displayed to Participant . . . . .	70
4.11	Question 3: Potential Partial Palette Based on Weights . . . . .	70
4.12	Question 4: Which Tools Would be Helpful? . . . . .	71
4.13	Question 4: Code Snippet Displayed to Participant . . . . .	73
4.14	Question 4: Potential Partial Palette Based on Weights . . . . .	73
4.15	Question 5: Which Tools Would be Helpful? . . . . .	74
4.16	Question 5: Code Snippet Displayed to Participant . . . . .	75
4.17	Question 5: Potential Partial Palette Based on Weights . . . . .	75
4.18	Question 6: Which Tools Would be Helpful? . . . . .	76
4.19	Question 6: Code Snippet Displayed to Participant . . . . .	78
4.20	Question 6: Potential Partial Palette Based on Weights . . . . .	78
4.21	Recommended Palette: Empty File . . . . .	87
4.22	Recommended Palette: Editing a Script Without Imported Libraries . . . . .	88
4.23	Recommended Palette: Editing a Script With Imported Libraries . . . . .	89
4.24	Recommended Palette: Editing a Class . . . . .	90
4.25	Recommended Palette: Error in the Code . . . . .	91

# Chapter 1

## Introduction

### 1.1 Motivation

Computers are ubiquitous in our lives, from devices we recognize as computers to the mobile computers we carry in our hands and pockets all the way down to all of our smart devices like thermostats, light switches, and even range hoods. This proliferation of computing devices requires skilled workers to create the software these devices require to operate. A 2021 “Beyond the Numbers” report by the U.S. Bureau of Labor Statistics [4] states that from 2019 to 2029 computer occupations in aggregate are projected to experience growth of 11.5%, about three times the average rate of job growth. Jobs for software developers are projected to grow 21.5%, computer and information research scientists are projected to grow at 15.4%, and a group that comprises of data scientists is experiencing explosive growth of 26.5%, albeit from a smaller starting pool of jobs. These in turn will generate around a half of a million new jobs by 2029 on top of the existing labor needs, which is only growing as people retire and transition out of the field. This clearly demonstrates a need for education and training to fill these new openings. According to the NSF 2022 state of science and engineering report [5], science and engineering degrees account for 27% of degrees awarded, and bachelor’s degrees account for nearly 70% of all science and engineering degrees awarded, which most of these were in agricultural, biological, or social sciences. Science and engineering master’s degrees increased the most in computer sciences and engineering, mostly driven by foreign students on visas. The report goes on to say the “performance of U.S. K–12 students in STEM has been stagnant” as measured by a lack of improvement in mathematics test scores for all students over more than a decade while scores for underrepresented minorities lag behind white and Asian students. Thus, the report argues,

the U.S. labor force for STEM jobs currently relies upon foreign talent.

There are many factors that contribute to, or prevent, students matriculating into and successfully completing STEM education in their university studies. Some scholars question the belief that there is a shortage of STEM workers [6, 7]; however, the aforementioned NSF report highlights demographic, socioeconomic, and geographic disparities in education and performance at the K–12 level for STEM education. A contributing factor is that K–12 educators often have little experience in STEM [8], especially in schools at the lower end of the socioeconomic strata and with high populations of minority students [5]. Programming is challenging for K–12 students to learn, especially when their teachers do not have a solid grounding in the concepts they are meant to teach. With teachers themselves having not learned computational thinking nor computing concepts in their formative years, it is difficult to expect them to obtain this knowledge from an undergraduate program designed to educate them how to broadly teach. Thus, these educators themselves become adult novices in programming and are expected to learn how to program, decipher which visual and text-based languages should be used for their instruction, and design courses to teach students how to program while guiding them through this transition from blocks to text languages. With the proliferation of both text-based and block-based languages only being outstripped by the proliferation of opinions about programming languages, it is a heavy ask of our K–12 teachers.

How to increase student matriculation into university-level STEM programs is a multi-pronged problem that requires many solutions from a diverse set of researchers. Some part of the solution includes the approaches and programming languages for teaching computer science. Weintrop and Wilensky [9] conclude that programming tool creators and educators will need to work together to facilitate solving this problem.

Papert was interested solving similar issues around computer science education; he created the Logo programming language [10] to help children relate mathematical concepts to their experiences and interests, which reinforced his constructionist educational philosophy [11]. Martin and Resnick expanded upon this work by creating LEGO/Logo to scaffold upon Papert’s constructionist, activity-oriented pedagogy [12]. The intent was for students to actively participate in their scientific development, which enabled them to learn how to think critically and systematically about problem solving. These pedagogical theories led these researchers to create these early visual languages to help students learn STEM concepts. Papert coined a term in his seminal work [10] for this, computational thinking, which is an apt term popularized by Wing [13] and now used throughout the research literature as a universal approach to solving problems. Its two main progenitors, Wing and Papert, conceptualized computational thinking from two different angles—respectively, thinking like a computer scientist regardless of profession and the result of constructionist



educational pedagogy that focuses on affective and social implications of computing.

Visual programming languages employ graphical elements, rather than solely text, to enable people to create programs. These graphical elements abstract away implementation details and syntax while allowing users to directly interact with the elements via a drag-and-drop interface from a menu onto an editing palette. These, in turn, allow people to create programs with arguably more visual scaffolding that might reduce some of the complexity of text-based languages. Most visual programming languages can be classified into one of four primary categories: form-based, diagram-based, icon-based, and block-based languages [14].

Block-based programming is receiving a growing amount of attention in the research literature and in classrooms due to the typical focus of using these languages for educational purposes. The intent behind block-based programming languages is to provide learners an introduction into computer science principles and programming in general, with the visual representation being easier to understand than text-based programming languages [15]. These languages typically employ bright colors, sprites (two-dimensional bitmap), sounds, and a heavy emphasis on actions to get children engaged into the idea of programming their own game. While these features make block languages playful and engaging, they lend to the critique of later-stage learners that block languages are too simplistic for all the possibilities of programming, even if they are unsure what those possibilities may be [16]. The primary question that is yet to be answered in the research literature is if block languages are better than text-based languages at teaching students programming concepts and, thus, fulfilling the future needs for professional computer occupations.

Unfortunately, this seems to have not yet borne out in the research. While many researchers seem to be trying to answer that question, it needs to be considered if the limited nature of block-based programming, and thus the limited duration of teaching computer science using these languages, prevents the deeper learning necessary to truly understand programming concepts in depth. In much of the research literature, studies are performed where block-based programming is used for a matter of mere weeks [2] [17] [18] [19]. While there are some rare exceptions of using block languages for a full course [20], block languages tend to not be used to teach multiple semesters nor for more advanced concepts, which may in turn hamper the scaffolding process, given the loss of the cumulative effect of time practicing writing in this modality [21]. Might this lack of depth and duration have contributed to the limited benefit currently seen with block-based languages?

Additionally, few programming languages, whether block-based or text-based, are accessible to people with disabilities, which may contribute to students with disabilities feeling uncomfortable with majoring

in computer science [22]. Ensuring these students are able to access the same learning opportunities as their peers is not just federal law. It is also how we ensure diverse perspectives are brought into computing fields, which in turn may help prevent computationally-reinforced societal decline from impactful vectors such as AI bias [23, 24].

Many computer science programs at the university level move from basic programming concepts in the first semester and then proceed to object-oriented programming, data structures, algorithms, and beyond in subsequent semesters. In order for block languages to be used, and effective, to scaffold the learning beyond the introduction of the basic introductory course, they would need to be capable of handling the complexities of teaching these concepts. Garcia et al. [25] created Block4DS to help address these deficiencies, and some preliminary findings were positive [26]. While they found no significant difference between learning outcomes between students who learned data structures with text-based or block-based modalities, all students had prior working knowledge of text-based programming yet none had experienced block-based languages prior to that course. This suggests that a thoughtfully constructed, scalable block-based language might be effective throughout a computer science program. Interestingly, in that study, female and English as a second language (ESL) students reported preferring the block-based language over text-based pseudocode. The primary deficiency of the study is it compared the instruction method—video based instruction given with pseudocode vs in the block language—via a concept-based pre- and post-test without students actually writing code in either modality, which may have impacted any potentially positive writing-to-learn effects [21].

Brennan et al. performed a study of existing Scratch projects from long-term Scratch programmers, and although these projects initially indicated student fluency as evidenced by existing computational concepts in the code, interviews revealed that students' descriptions sometimes demonstrated significant conceptual gaps, since they borrowed code from other projects and could not explain how the code worked [27]. This effectively demonstrates the need for a curriculum that can scaffold learning experiences with algorithmic computational problem solving to provide a strong foundation for success in future programming experiences [28], and part of that may be having the proper block languages to take learners through more than just the first few weeks of their programming courses. While there is an issue with the transfer of learning at this stage, it is not yet known why this occurs. Bangert-Drowns et al. [21] posit that there may be interference with the learning and writing relationship when students transition to new subject-specific writing forms in general education, which could potentially equate to transitioning from block-based to text-based modalities in programming. It could also be that the complexity of the curriculum at the stage

where the transition occurs is so low that the starting modality would never have an effect.

Despite Whitley and Blackwell’s findings that professionals expect visual programming languages to be less powerful, less readable, and less enjoyable to use than text-based languages [29], today there are professionally-used programming languages that use visualization in some form, which indicates that it is possible to entice professional programmers to use visual programming. For example, Unreal engine employs Blueprints [30], which is a highly visual programming environment, albeit different from blocks, as a primary programming modality. It is an interesting point to consider what makes some visual languages acceptable or unacceptable in the eyes of professional programmers. Is it purely based on stigma or popularity, or is it more based on the capabilities and attributes of each language?

The perspectives and opinions of software developers matter significantly in research about software engineering [31], but while studies have been done with professionals using visual programming [32] [29] [33], there is scant research specifically on block languages used by professionals. The experience and knowledge of software practitioners is crucial to validate assumptions and evaluate the tools, techniques, and methods software engineers utilize, and we know professional programmers do use visual programming languages. Thus, the research should assess how professionals evaluate the tools, techniques, and methods used to train the future generation of computer specialists. This could potentially generate the next generation of block languages that could be beneficial beyond the first few weeks of programming education, which could potentially be scaled to use broadly in professional programming contexts.

A radical concept to consider is if it would be possible to have a single programming language to cover programming needs from educational inception through a professional career. Much of the research about block languages revolves around the challenges of transitioning learners from block-based to text-based languages. If such a language could be devised that is as instructional for young learners as it is powerful for sophisticated computer scientists, then that transitional friction could be erased since that modality switch would no longer occur. Additionally, students would experience the positive effects of writing-to-learn over a longer period of time to accumulate more depth of experience. With this in mind, we should endeavor to discover what attributes of visual programming languages professionals find to be beneficial versus detrimental.

Could investing in introducing more powerful capabilities in block languages help them be used in subsequent courses and potentially used in professional development? What aspects of block-based programming might be beneficial to professionals, and by breaking down any barriers to using block languages, could we see professionals begin to adopt block-based programming at scale? In the first study, we address how

the visual attributes of block-based languages are perceived by professional programmers and computer science students alike. These attributes may enhance or deter student learning outcomes. Likewise, they may contribute to how professional programmers perceive a language’s capability and usefulness in their own projects. In the second study, we investigate how a structured IDE might improve the usefulness of a block-based programming environment. Factors we analyze include the context-aware suggestions for what types of blocks and structures to present to the user, error detection, error alerts, context-sensitive suggestions for variable use, and determining the prominence of displaying links to documentation. With the results of these studies, we can make a recommendation how to design a future block-based programming environment.

## 1.2 Research Questions

With the importance of this topic and the breadth of application, there are a multitude of potential research questions. Within the scope of this study, the broad research questions we plan to study are as follows:

1. What attributes of block programming languages and environments do professionals find to be beneficial versus detrimental?
2. What aspects of block-based programming might be beneficial to both learners and professionals?
3. What capabilities could be added to a block language and environment to ease the transitional friction for learners to become professionals?

## 1.3 Contributions

In the first study in chapter 3, we present the results of how professional software developers perceive attributes of block languages to determine which visual attributes are helpful versus harmful. We then compare these measurements to results we obtained from a pool of university level computer science students. These results in turn provide insights into what similarities and differences there are between the two groups: professionals and learners. A total of 87 participants took part in the entire study: 30 professionals and 57 learners. In the study we presented a small program to participants to have them solve a programming question on a read-only basis. After each problem, we collected Likert scaled feedback about how each visual attribute helped or harmed the participant’s ability to solve the problem.

The survey instrument displayed a small program that was syntactically the same for all participants, but the visual attributes were varied. When a participant was assigned to a particular set of attributes, they were then given the same attributes for all questions in the survey. The motive of varying the attributes was to be able to test how certain primary attributes affected the participants' perceptions of block languages in general. Particularly, we aimed to see how altering the color and layout of the small program affected the participants' perception of an array of attributes. We also aimed to determine if non-code helpers, such as line numbers and comments, were perceived as beneficial even to small, simple programs. The results of the survey were analyzed quantitatively.

In the second study in chapter 4, we analyzed potential context-aware capabilities of an integrated development environment (IDE) for a block-based language. In particular, we sought to discover how such an IDE could be context-aware enough to provide helpful suggestions and insights when programming. We compared the results of professional programmers' opinions with university level computer science students. A total of 123 participants completed the study: 21 professionals and 102 novices. In this study, we presented several different scenarios where the participant was asked to imagine a specific programming challenge. The participant was given a small snippet of code and an indicator where the cursor was and then asked to categorize 16 tools into helpful versus not helpful, for that specific scenario. Then the participant was asked to rank the helpful items from most helpful, next most helpful, and on down the line.

The survey instrument displayed the same questions to all participants without any changes or variations. The intent of the single-group design was to determine if there was any preferential deviation between tools and if there was a difference in tool preferences between professionals and novices. Specifically, we aimed to find if some tools are broadly more important than other tools, if certain tools are more important in various scenarios, and if professionals and novices view the tools' usefulness differently. The results of the study were analyzed quantitatively.

### **1.3.1 Summary of Results**

The first study quantifies what visual attributes professionals and learners stated to be helpful or harmful for completing tasks. Results show that colors and color categorization are beneficial for both learners and professionals. Color is beneficial for people without visual impairment and is critical to assess for those with visual impairment. Determining how to use colors and categorize colors would be a beneficial step forward for a future work. When analyzing the results of a horizontal, dispersed block layout, this format is shown to be harmful to understanding for learners and professionals. The evidence shows that

allowing blocks to be randomly placed on the palette, thus forcing horizontal scroll, is detrimental to both learners and professionals. Distinctive block shapes were beneficial to task completion across the board for learners, but results were split for professionals. While we do not have significant measures of correctness, the benefits of block shapes were circumspect for professionals. This attribute needs to be studied further, especially considering how it could impact visually impaired learners. Results additionally show that the lack of comments and line numbers are harmful for both learners and professionals alike. The evidence demonstrates that both groups would find the ability to include comments and reference line numbers to be beneficial.

The second study quantifies what individual tools, or features, professionals and novices found to be beneficial in various scenarios. Professionals and novices categorized and ranked tools similarly, so there was no significant difference in experience groups. Results show that there are some tools, such as “Example Code,” that are beneficial in a broad range of programming scenarios, while other tools, like “Source Control,” are not as beneficial to have in the IDE. Other tools are beneficial to display consistently, such as “Blocks for Conditionals” or “Blocks for Controls,” whereas there are some specific use case tools, like “Blocks to Import Additional Libraries” that appear to provide little benefit. Overall, the results indicate that a context-aware palette of tools appears to be a beneficial addition to a block-based IDE. As discussed with Laxmi Gewali [34], this dissertation is at the intersection of computer science education, human computer interaction, and programming languages.

# Chapter 2

## Review of Literature

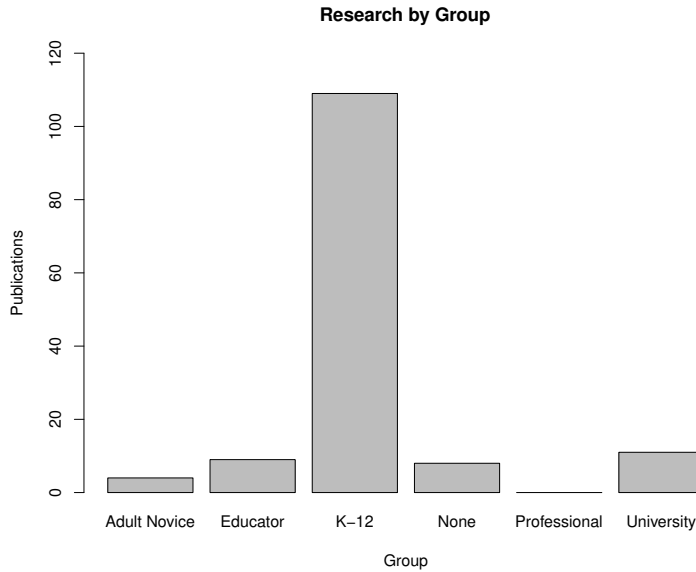
### 2.1 Overview

The goal of this review is to investigate the role of block-based programming languages in computer science education, particularly how transitional friction plays a part of reducing the value of starting the early educational stages in block languages instead of text. We provide an overview of the history of block languages and they fit into the landscape of visual programming languages. A review of benefits and drawbacks to block languages examines why block languages are utilized, albeit on a limited basis. Through this review, we discover that important groups are underrepresented in the research about block languages: people with disabilities as a whole, adults who only need to perform ad-hoc programming activities, K–12 teachers (who need educating on computer science fundamentals), and professional programmers.

### 2.2 Literature Distribution

For this research, we analyzed 403 research papers on visual programming languages. From that, we categorized a sampling of 141 research papers related to block-based programming, with the primary goal being to find gaps in the research where certain groups have been under-studied. Additionally, we wanted to determine which visual elements of block-based programming help or hinder use. While this review is not comprehensive of every research paper about block-based programming, it does cover a broad swath of the current literature. Papers were chosen for classification by choosing a sampling of these top 403 most relevant articles from the ACM's, IEEE's, and ScienceDirect's digital libraries. We classified the papers in figure 2.1 across age-experience groups: K–12 students, university students, adult novices, computer science

Figure 2.1: Research By Group



educators, professional software engineers, and a “none” category, which typically meant it presented a new block-based tool without additional research done on it. With this classification, we found that 77.3% of papers studied the effects of block-based languages on K–12 student participants, whereas 8.5% of research was conducted with university students as subjects, and there were no papers that performed any type of block-based programming research on industry-level computer scientists.

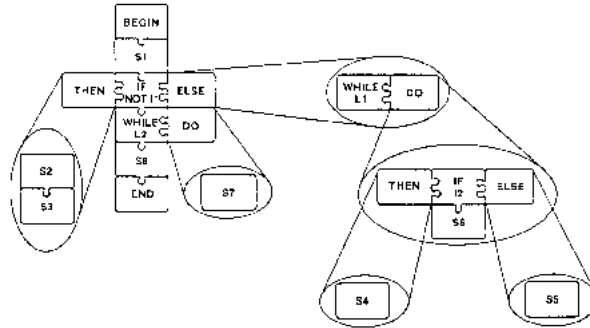
Additionally, less than 3% were conducted on adult novices and 6.3% with educators as the participants of the study. If we are to ensure that everyone has access to computational fields, we need to ensure we understand how adults can be better served by block-based languages, whether they are learning to program for themselves or they are educators who are planning to teach our students. Thus, the research body should be expanded to quantify the teaching and learning capabilities that block-based languages afford or deter.

### 2.3 Visual Programming Languages

The true roots of visual programming languages are debated. While some consider the first visual languages to have started in the 1980s with Pict [35], Blox 2.2 [1], LabVIEW [36], and Logo [10], which is purely text-based with visual components, others [37, 38] go back as far as Jackson’s [39] work on visual structured programming or Sutherland’s [40] dissertation work on a graphical associative programming



Figure 2.2: Glinert’s Blox, an Early Visual Programming Language [1]

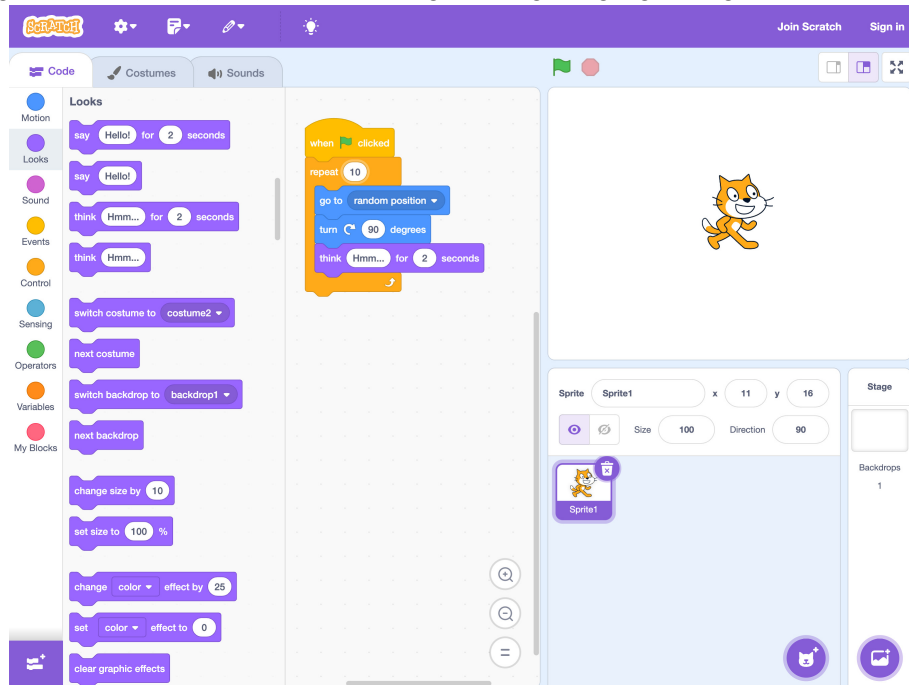


language in 1966 as the original, albeit rudimentary, visual programming languages. Logo, which was started in the 1960s [41], long before adding visual components, is the progenitor of over 300 programming languages [42], many of which are visual languages such as Scratch 2.3 and LogoBlocks, a true visual programming implementation of Logo [43]. Myers [44] asserted that visual languages include flow charts or graphical languages that are processed in multi-dimensions, not in a one-dimensional stream like conventional text-based languages.

With humans being so highly visual [45], it comes as no surprise that visual languages have been in development and use for so long. Visual programming languages attempt to convert the complexity of programming syntax to a graphical representation. This visual element serves multiple purposes; at a minimum they act as a memory aid, simplify complex character sequences into something more intelligible, and provide a more appealing layout. Arguably, visual languages are more approachable to first-time learners; unfortunately, this visual representation is not easier for everyone. It is understandable, although not excusable, that people who are blind or low-vision are often not considered in the context of using visual languages. While that is in progress of changing [46, 47, 48], these tools have not yet penetrated the list of top educational programming languages.

Visual programming language implementations run the gamut of education [49, 50], robotics [51], video game development [30, 52], and, more specifically, block-based programming languages have been created for specialty concepts such as robotics [53], data science [54, 55], Internet of Things (IoT) [56], to aid in computational thinking in archival science [57], parallel programming [58, 59], notebooks [60], and IDE-like workbenches [61] for block languages. Visual programming languages can be classified into one of four primary categories: form-based, diagram-based, icon-based, and block-based languages [14].

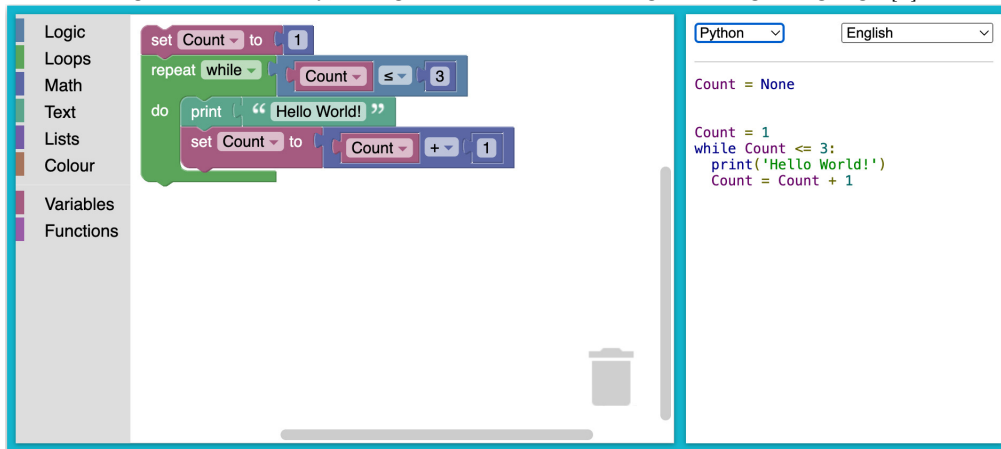
Figure 2.3: Scratch, a Block-Based Programming Language Designed for Children [2]



## 2.4 What Are Block-Based Programming Languages?

Block-based programming languages are a subset of visual programming languages. Block languages use a puzzle-piece metaphor approach to programming [2]; blocks are similar to puzzle pieces in that they can interlock with each other to create programs. In these environments, users drag-and-drop blocks that represent program components, such as variables, conditions, expressions, and statements, onto an editing palette. Some languages only allow blocks to be interlocked vertically, some only allow horizontal interlocking, and many employ both vertical and horizontal interlocking to construct program syntax. These blocks help to show semantic groupings of code, such as nesting a group of commands inside of a while loop. Some of these languages allow multiple grouping of blocks disaggregated from each other, while others require all blocks to be connected. Many block-based languages tend to not require left-alignment to a margin as a text-based programming language would in a traditional IDE. This means blocks could be dispersed on various parts of the canvas, called spatial layout, which potentially would allow more code to be seen on a single screen. Blocks could be broken into logical chunks, such as functions or imports, and placed horizontally as desired. This would affect both the potential scrolling directions on the canvas and how the blocks can be arranged. It is often unclear how programs will execute when the disconnected

Figure 2.4: Blockly, Google’s Block-Based Programming Language [3]



blocks are dispersed across the canvas, and each language implementation seems to have its own way of dealing with that layout.

Visual cues, like color and shape of the blocks, help provide guidance on how the blocks can be snapped together to form a logical grouping. Blocks that can interlock together usually will “magnetically snap” in place with each other to ease construction of programmatic phrases. This also means blocks that cannot interlock cannot be accidentally put in a place that would make the code syntactically invalid. The environment prevents any groupings that are syntactically invalid. Some block languages only minimally allow typing, limiting it to just filling in the blanks where variable names and numerical and string literals go, whereas others allow for broader use of the keyboard to provide more input flexibility to the programmer. Utilizing blocks removes the need for complex syntactic elements, such as parentheses, braces, and semicolons to delineate code sections. Together, these arguably diminish syntactical precision as a barrier to programming [62].

These environments share characteristics that are thought to aid in learning programming. In many environments targeting children, programmers can create games, stories, and art with highly colorful blocks that focus around actions like movement and playing sounds. Several distinguishing features of block languages comprise of palette of commonly-used blocks for easy access and recall, using colors and shapes to delineate programmatic concepts and placement, and employing natural-language descriptions of the block’s purpose rather than esoteric syntax. Design decisions were made throughout the process of creating these languages, which should be evaluated to determine if they are beneficial or helpful, such as the limited palette of blocks, horizontal scrolling, colors that in some cases do not meet WCAG guidelines [63], access

to documentation, and more. Even with historically mixed evidence for and against visual programming languages [29, 64, 65, 32], they have proliferated. A small selection of currently successful block languages include Scratch [49], Snap! [66], Alice [50], Blockly 2.4 [67], AppLab by Code.org [68], Makecode [69], and Quorum [47].

The Scratch programming language is a seminal work among block-based languages. Resnik et al. [49] state that digital fluency requires “not just the ability to chat, browse, and interact but also the ability to design, create, and invent with new media,” which requires learning some type of programming. They state that mastering programming is difficult because programming is often introduced with activities that are not connected to someone’s interest or experience, as well as in contexts where novices could not get guidance when things go wrong, nor encouragement to dive deeper when things are working. Papert discusses the concepts of low floor, high ceiling, and wide walls [10]. He defines a low floor as the ease of starting to work with a new technology, while a high ceiling provides for increased sophistication over time. Wide walls is defined as encompassing as many use-cases as possible. Across the literature, core concerns are to make Scratch tinkerable, meaningful, and social, which some term as “breadth,” or range of distinct features used, and “depth,” which they define as the amount of features that are used [70]. In teaching students scientific concepts with LEGO/Logo, Martin and Resnick [12] equated this type of student learning to being both scientists and inventors. Students learned how things *are* by creating and testing theories while simultaneously learning how things *could be* by creating new programs and brick-constructions.

## 2.5 Why Block Languages?

The research literature demonstrates that students without prior programming experience are at a reduced likelihood of being successful in completing computing courses, and it is especially true for female students and those from underrepresented groups [71, 72, 73, 74], which often causes high dropout rates in the first course for these students. Bui et al. [75] found that students without prior experience who persist through introductory computer science (CS1) courses go on to experience performance gaps in second level computer science (CS2) courses, which in turn has a knock on effect for lower performance in upper level courses [76], again affecting underrepresented groups and females to a higher degree. Bui et al. [75] discover there is no difference in CS1 and CS2 performance between students with formal or informal prior experience in programming as long as there was some experience prior to entering CS1, so the critical factor is ensuring all students, especially those in less advantaged demographics, get experience in computing and

programming prior to entering computer science programs.

There is a growing body of research discussing how novices, especially in K–12, can best begin learning to program using block-based languages [77, 50, 78, 79, 80]. These languages are intended to entice students to experiment and play as a method of learning programming skills and computational thinking. It is thought that block-based programming makes it easier for novices to begin programming due to visual cues, mitigating syntax errors, presenting the available set of commands, removing typing challenges, and the use of natural language. Maloney’s stated [81] intention behind why Scratch was created is simply that it “encouraged young people to learn through exploration and peer sharing, with less focus on direct instruction than other programming languages.” Cooper’s [82] viewpoint on Alice is that it engages students to program within a context to tell stories, which forces students to learn programming by engaging in the process of writing code.

Weintrop et al. posited reasons why block-based programming might be easier than text-based programming [19]. When we focus solely on readability, the attributes consist of the shape and visual layout, memory aids by way of color categorization, and the perception that blocks are easier to understand due to them being more like human-readable English (although approximating English is debatable depending on the block language used). While attributes like color and shape do not change the meaning of blocks, they may aid in comprehension, task completion speed, and recall. In block languages, colors are used to categorize blocks into similar functions, so control structures, output, and operator block categories are colored differently from each other. This differs from a traditional IDE where all keywords are the same color, regardless of type; for example, “int”, “for”, “if”, and “public static void” are all the same color in traditional text-based IDEs for Java even though they differ in function. Additionally, the shape of certain blocks, such as looping structures, convey that all the code inside of them is grouped together, which simplifies the concept of scope. It is claimed [81, 15] that these visual attributes work in concert, potentially aiding in comprehension and learning programming. Price [83] reported that students complete programming challenges faster using block languages, although it is not empirically shown if that effect was due to these attributes and how they were designed.

Students’ interest in continuing to take computer science courses is, as expected, critical to success in completing computer science programs. Armoni et al. [79] concluded that teachers reported a doubling of students enrolling in additional computer science courses when students took an introductory course in a block language first, which was attributed to their positive attitudes toward programming after the block-based course. Weintrop and Wilensky [84] found that after a five-week programming course, students who

used a block-based modality reported an increased interest in taking additional computer science courses whereas those who took a text-based modality reported lower interest in subsequent courses. The caveat to this is selection bias, since these students being studied are the ones who elected to take an initial programming course. In many other cases [80, 85, 86], interest in additional computing courses started high and remained relatively high by the end of the course.

## 2.6 Challenges with Blocks

While block languages afford many benefits to learners, there are challenges to their usefulness in education.

### 2.6.1 Self-Efficacy & Learning

The efficacy of visual languages has been long debated [29]. Dwyer et al. [87] posit that reading code in block-based environments to be complex. When any block-based program grows to a certain extent, there are many components to analyze, which attributed to the complexity. Fundamentally, when code becomes more complex and longer, it does not become easier to read, even if it is a block-based language. Additionally, the claims that blocks syntax, or even any programming language syntax, are easier to understand than text due to any specific syntactic word choice, is a difficult claim to make [88, 89] and should be further studied in future works. From some students' perspectives, block languages are “less powerful,” slower to author, more verbose, and “inauthentic” relative to text-based languages such as Java [19]. These issues juxtapose the concepts that despite blocks trying to simplify programming concepts, they might complicate it, while simultaneously being viewed as far too simple to be useful to some.

Weintrop [16] investigated code-reading comprehension for variables, iterative logic, conditional logic, and procedures, but found no difference in program comprehension between text-based and block-based languages, which extends the need to do further research regarding Price's [83] findings. In another study [90], students demonstrated the ability to understand what a construct *does* but not *how and when* to use that construct when solving problems. This presents a problem for constructionists who are champions for self-directed students teaching programming principles to themselves. While some researchers [91] suggest putting the onus on students to be more engaged in their own learning by pure discovery, this is a difficult expectation to place on K-9 students, aka children, who are learning programming with Scratch or other visual programming languages. Children are not always the most autodidactically rigorous, and this often is even more true of those from disadvantaged groups. In cases of self-directed learning, we tend to see

code quality [92, 93, 94] and comprehension [90, 95] issues.

In total, these challenge Papert’s and Resnik’s assertion of tinkerability and students leading their own learning activities. While that certainly works in some cases, we must also remember the selection bias inherent in their findings. While block languages do appear to be helping us move toward improved computer science learning, we cannot place an undue burden of that onto the learning environment nor the learner.

### 2.6.2 Accessibility

A major challenge for block-based languages, and really any visual language, is accessibility. A number of researchers working to make block-based languages accessible to visually impaired students [46, 96, 47, 97, 48, 98], neurodiverse students [99, 100], and students with physical impairments [101]. Unfortunately, these languages are not widely used throughout our society and educational system, which prevents many students with disabilities from participating in computer science learning activities with their peers. At this time it does not appear to that the Scratch team, LEGO corporation, or other prominent block language developers are addressing these accessibility issues, but it would be beneficial if they did considering the moral, ethical, and legal implications. When the Computer Science for All initiative was created, the “All” portion of it was intended to be inclusive of *everyone*, not just those without disabilities.

## 2.7 Transitioning from Blocks to Text

How to transition novices from basic block-based programming into a more fully-featured text-based programming language is frequently discussed in the literature. Studies often start with novices of various ages to determine how to ease this so-called, “transfer of learning.” Unfortunately, with due deference to the importance of transfer of learning research, studies showed that learning advantages may cease to exist after only 10 weeks, meaning no additional impact [102]. If this transitional friction is true, this could bring into doubt transfer of learning’s real impact if not generalized. Alrubaye et al. [103] gathered preliminary results that indicates there may be a positive transfer of learning when moving from hybrid block/text environments to a text-based environment, but it should be noted that it was a small scale study with few participants and conducted over a two-hour period rather than an actual educational course. To really understand if this “transfer of learning” is real and sustained, a longer-term study would need to replicate these findings.

Further, novices generally do not pursue adequate depth and breadth of their own volition [104], meaning they tend not to learn advanced programming skills on their own, without a dedicated curriculum. Kurland and Pea reported that young students with a year of self-guided, discovery-style programming experience in LOGO were able to write and interpret short, simple programs, but they had difficulty creating programs involving fundamental programming concepts and had many incorrect perceptions of how programs work [105]. Moors et al. [95] concluded that self-guided exploration in block languages contributes to poor programming practices and a lack of understanding programming fundamentals. Poorly learned fundamentals and practices can exacerbate problems in the transition from block languages to a more difficult text-based modality.

While there may not be a difference in capabilities when transitioning from block-based to text-based languages, there may be a deleterious effect on confidence. In studying the effect of transitioning from Alice to Java or C++, Powers et al. found that students were overwhelmed by all the syntax, and they were discouraged when their programs would not compile, which led to both strong and weak students losing confidence in their programming capabilities [106]. Students found Alice and Scratch, in another study that compared Scratch to text-based programming [80], to not resemble “true” programming, and students that programmed in text-based languages had higher confidence in their skills. Sometimes words like “authentic” or “inauthentic” were used to describe such observations by students. Loss of confidence has shown to be a reason that affects student drop out rates in computer science programs [107], which makes this a precarious transition point in a learner’s journey.

Conversely, Bau endeavored to reverse the loss in confidence with Droplets and Pencil code [108, 109]. Weintrop and Wilensky [110] found no significant difference in student’s confidence levels between block-based and text-based programming when using Pencil.cc, a customized Pencil Code environment that allows students to program in either text or blocks. Krafft et al. [111] introduced block-based programming with Scratch to university students and staff without a background in computer science. While the researchers did not see an effect on novices’ abilities, they did see positive effects on their identities as novices and their motivation to continue learning. The implication is that carefully designed block languages that are not too distinctively different from text-based programming may help prevent loss of confidence for novices in the transition from blocks to text. Furthermore, Armoni et al. [79] reported increased enrollment in computer science courses and higher levels of motivation and self-efficacy in students who took a five-month programming class exclusively in Scratch before transitioning to a text-based programming course, which calls the loss of confidence into question.



Weintrop et al. posited that future programming for non-computer science people could be done entirely in a block-based language if it was carefully considered and designed [102]. This suggests that all programming tasks could be performed successfully in a block-based environment, while being understood and used at least as successfully as in a text-based programming environment. The caveat here is that for a block language to aid in the transition from novice to professional, it would be beneficial for it to be a structured editor built on top of an existing language, rather than a stand-alone visual programming language [112].

## 2.8 Computational Thinking

Computational thinking, a term loaded with multiple connotations by an array of stakeholders, is summarized by Lodi [113] as “a form of thinking for solving problems by expressing the solution in a way that can be automatically carried out by an (external) processing agent.” In essence, it is approaching and solving problems with the same mindset as a computer scientist. Brennan and Resnick [27] created a framework of three key dimensions for computational thinking: computational concepts (e.g. sequences, loops, events, etc), computational practices (incremental and iterative, testing and debugging, etc), and computational perspectives (expressing, connecting, questioning, etc). Through the research of Zhang and Nouri [114], which expanded upon Brennan and Resnick’s work, all computational thinking skills classified by these researchers were able to be taught with a single visual programming language—Scratch [49]—even though insofar it still rarely scales beyond a few weeks of instruction in a K–12 classroom. Many researchers have expanded on the initial frameworks for computational thinking [114] with the intent to help educators and society determine what to teach and what can be learned, questions that are crucial to teaching any subject.

The considerations for using block languages, or any visual languages, is critical to address for the advancement of computational thinking in K–12 and higher education. A primary intent of block languages is to enable learners with no prior experience to immediately begin experimenting with programming [81], the basis of computational thinking. While Wing [115] initially believed that it was an insurmountable task to get computational thinking into K–12, her work along with those of countless others is making that dream into a reality despite researchers and educators alike wrestling with the meaning and scope of computational thinking in practice [116]. The approachability of block languages is fundamental to introducing computational thinking pedagogy into K–12 and higher education, and computational thinking

concepts are starting to be included in standards such as the Common Core Mathematics and the Next Generation Science Standards.

## 2.9 Blocks for Educating Educators

A particular area of interest in easing transitional friction is training educators at the K–12 level. There cannot be an expectation of students at the K–12 level learning programming and computational thinking broadly if our educators are not properly prepared to teach these subjects. While many workshops and outreach programs [117, 118, 119] have been created to help prepare teachers, there is a degree of self-selection bias in those who choose to attend these workshops. Computational thinking competencies are not often included in the bachelors degree programs that most K–12 educators complete [120]; they are mostly relegated to specialty workshops or specialized university certificate programs and degrees.

Programming in block-based languages, and the transition from them to text, is only one aspect of transitional friction. Our K–12 educators must be first taught computational thinking and programming, then how to design pedagogy about it, and then understand both that this friction exists and where in order to smooth it out. They are also hampered by the same limitations that prevent developers from more broadly using block languages such as enabling collaboration, version control, available libraries to extend code capabilities, robust debugging capabilities, and automated testing. This is a large ask of individual educators to address without broad and deep support from multiple societal angles. One potential solution that could ease the educational burden as well as remove the transitional friction might be making block languages themselves able to scale from children to professional developers.

The research literature contains many other block-based programming studies directed at educators, which range from curriculum design and tools [121, 122, 96, 123, 124] to grading and rubrics [125, 126].

## 2.10 Blocks Beyond Formal Education

Shepherd et al. [127] discussed using a block-based language to address difficulties with programming basic industrial robots. Programming languages for industrial robots are difficult to use; they have esoteric naming conventions and require the use of buttons on a control pad, in addition to the language, to function properly. In fact, the research literature is focused predominantly on K–12 novices; adults outside of the university setting are only studied when they are novices outside of the typical computer science and programming realm. Various studies [128, 129, 130, 131] have addressed professional adult novices learning

how to program for business purposes. In these, the focus was not on educating them in traditional computer science principles; rather it was making programming tasks easier for adult novices who do not have a professional software engineer to help solve small business needs or for programming manufacturing equipment.

## 2.11 Professional Software Engineers

When broadening the scope to include empirical studies using any type of visual programming language, in one study, professional programmers noted that while visual programming might improve the ease of writing code, they believed visual programming would be significantly less powerful [32]. This study was an early indicator of professionals' reasoning for rejecting many types of visual programming languages. A related study conducted broadly on visual programming showed that visuals provide explicit, organized information, and using them helps improve correctness, speed, or both [29]. Conversely, Bragdon et al. found that professionals using Code Bubbles, a diagram-based language where snippets of code from multiple files in a project laid out on a canvas (which seem less organized due to a disperse layout), helped improved the time to complete tasks and correctness [33], although they had arrows and other constructs to aid in organization. Since navigation was only a small part of the time savings, Bragdon et al. hypothesized that professionals were able to shift focus quickly and offload their working memory because code was kept on the screen in front of them. Unreal Blueprints, a diagram-based language, also make use of horizontal layout and their tools are adopted heavily by game designers at the professional level. Diagram-based visual languages utilize various techniques (such as lines and arrows) to show code flow, to mitigate having a disperse layout.

Prior studies demonstrated that program comprehension and its related tasks took over half of the time spent during the software development and maintenance process [132, 133]. Developers assimilate a program's purpose primarily through reading the comments, but writing comments is often sparsely applied in software development, which can make program comprehension more difficult and time consuming [134]. This sparse "documentation" via comments means that developers must spend more time analyzing the actual code itself to understand what it is they are maintaining. This is problematic in light of studies that demonstrate that reading code in block languages is slower than text languages [29, 87]. The slowdown in developer productivity could lead to broad economic impact if languages like these were adopted.

Despite any challenges, assumed or observed, the time, money, and proliferation of research poured into

block languages begs the question of why there is a dearth of research addressing how trained computer science professionals could use block-based languages. Clearly, visual languages, and blocks specifically, could help with task speed, correctness, and focus. Even with a powerful visual programming language cousin to blocks, Unreal Blueprints, there is little research on usability by professional computer scientists and programmers [135, 136] despite the popularity of the language. Certainly, power of the language, authoring speed, and limited functionality [19] for current block-based programming languages are considerations that could hinder professionals, but again, this does not seem to hamper Blueprints from being the go-to choice for game development. Having a block-based language that can scale from K–12 education through professional project development would mitigate any learning transfer issues.

Weber [112] posits that minimal keyboard support (block-based languages often require using a mouse to drag and drop code) might be the primary reason why block languages are not used more widely by professionals. Additionally, lacking the ability to scale, collaboration, version control, and widely-available libraries and tools [62] currently prevent block-based languages from being used at production scale, so potentially this lack of maturity of block languages is the primary hindrance that eventually can be overcome. But also, might less empirical factors prevent professionals from using block-based languages? Perhaps text-based languages *feel* more like “real programming” because that is the way it has “always been” done by “real software developers.” Maybe the colorful blocks look like a child’s play-toy rather than a serious tool for work. Regardless of the reasons, empirical or emotional, we may garner insights into improving both programming learnability and professional programming practice by studying how professional computer scientists handle block-based programming languages.

## 2.12 The State of Block Editors and IDEs

Programs must be created inside of an environment that supports syntax construction for the language being written. In the case of text-based languages, this could be as simple as any text editor or complex like an integrated development environment (IDE) with robust features. To garner the benefits of visual languages, a number of researchers [137, 138, 139] have created visual IDEs or overlays for text-based languages.

Visual languages typically have their own environment where everything is contained. Maloney et al. [81] line out several of the design decisions made for the Scratch environment: single-window interface, being “live” and tinkerable, making execution visible, no error messages, making data visible and concrete,

and minimizing the set of commands. These design decisions were intended to entice self-directed learning and exploration in an easily navigable way. After performing a study on a Scratch-based language on fourth graders, Dwyer et al. [87] concluded that features in the interface created sources of both information and misinformation for users. Because of the density of visual information on the screen, some of which is quite subtle, students overlooked relevant features. Moreover, students believed some features were important and conveyed certain information that they actually did not, which created misinformation in the students' understanding of how programs worked.

Lin and Weintrop [140] analyzed 46 block-based programming environments and classified them into four categories: blocks-only, one-way transition, dual-modality, and hybrid. Of particular note is the hybrid environment, which blends text-based and block-based features into a keyboard-driven block-based programming environment. While this category is still nascent, students in the study who used the hybrid environment reported high satisfaction and low frustration while also performing tasks significantly faster than a group using Java to perform the same tasks. This suggests there is an avenue for a block-based programming language and IDE to scale from an educational context through to the professional sphere.

By gathering feedback from both novices and professionals in the same study about what visual attributes and context-aware suggestions are helpful or harmful, we may see patterns emerge to help guide the future of block-based programming language and IDE evolution.

## Chapter 3

# Study 1: Assessing Visual Attributes’ Role in Readability and Comprehension of Block-Based Programs

### 3.1 Methods

#### 3.1.1 Hypotheses

In pursuit of assessing developers’ perceptions of visual attributes that are inherent to block languages, we formulated the following null hypotheses:

1. There will be no difference in preference or correctness between groups given the block colors or grayscale treatments.
2. There will be no difference in preference or correctness between groups given the single-aligned vertical block layout arrangement or the spatial (dispersed, horizontal) layout.
3. There will be no difference in preference for block shapes regardless of treatment group or experience.
4. There will be no preference for the lack of comments and line numbers.

### **3.1.2 Inclusion and Exclusion**

We included preliminary demographic questions to determine each participant’s highest level of completed education, current level if still in college, if they are currently or have ever been employed to write code professionally, and what programming languages they have used to write code. None of these questions excluded or included a participant in the study, nor did it influence to which group (see table 3.1) they are assigned. These data were collected solely for the purpose of determining the effect of the particular task upon each level of experience: professional or novice.

### **3.1.3 Participant Characteristics**

We recruited both university students and professional programmers to help understand the potential differences. University students were recruited from the authors’ institution’s undergraduate and graduate-level computer science program, and professional programmers were solicited on LinkedIn and via direct email. If students had professional experience, they were included in the professional category.

### **3.1.4 Sampling Procedures**

Participants were recruited from four university level undergraduate courses at the authors’ institution. Additionally, participants were solicited through professional networks such as LinkedIn and through the corresponding author’s network. Participants were presented the option to participate and self-selected into the study. There were a total of 87 participants, 30 of which were classified as professionals and 57 of which were classified as novices. Classification was done during the demographics questions, one regarding if the participant was currently or had ever previously been employed to write code as all or part of their job function. After this, participants were then randomly assigned to groups by the online testing system, which resulted in both professionals and novices being randomly assigned to each of the four groups (see table 3.1). Due to this random assignment and experience level classification, the number of professionals were not evenly distributed to the groups.

### **3.1.5 Sample Size, Power, and Precision**

As we are unaware of an existing study on the same survey, there was no direct way to calculate our required sample size, power, or precision. As such, we undertook a series of pilot studies to provide critical feedback on the study from participants. All participants that participated in these pilots were separate

Table 3.1: Groups

Group	Block Coloring	Scroll Direction
A	Color	Vertical
B	Color	Horizontal
C	Grayscale	Vertical
D	Grayscale	Horizontal

from those reported in this study. In these pilots, we started with a small sample size, then roughly doubled from one major pilot version to the next. The data from these pilots, which is ultimately consistent with the data reported here, is not directly comparable due to changes in questions and experimental design, but we point it out here as it was an important part of our process in creating our instrument.

### 3.1.6 Measures and Covariates

The instrument asked a programming question about a small program in the block-based language Blockly [141] and then asked to rate nine attributes. A Likert scale was used to rate each attribute independently. This progression was repeated for seven programming questions, which forms the basis of a repeated measures design. The primary distinctions among groups are for group differences as shown in table 3.1. The direction of scrolling and alignment (dispersed/horizontal only vs in-line/vertical only) and coloring of the blocks (full color vs grayscale) form the two-by-two between-subjects design. Participants were assigned into “novice” vs “professional” categories based on self-reported demographics questions about their prior work experience, which transforms the study to a two-by-two-by-two between-subjects repeated measures design when comparing the groups to each other.

After answering each programming task, the participants were asked to select how helpful or harmful each visual indicator as shown in table 3.2 was in answering the previous question. Participants were presented a Likert scale in the form of sliders ranging from one to seven, with one meaning “extremely harmful” and seven meaning “extremely helpful.” The midway point of four (“neither helpful nor harmful”) was the default position, but participants were required to at least click on each Likert slider in order to proceed. The “other” attribute was not analyzed for this work, since the participants were able to write in their own free-form text in this field. Write-in results from the “other” attribute may be analyzed for future research directions.

Task timing and correctness are also considered in the analysis. Due to the uneven distribution of professionals and few participants in two groups, it is difficult to draw strong conclusions from these



Table 3.2: Attributes Assessed

Attr #	Attr Description
1	Color of the block
2	Shape of the block
3	Arrangement of the blocks
4	Scrolling
5	Text, punctuation, or symbols
6	Lack of comments
7	Lack of line numbers
8	Spacing
9	Dropdown arrow next to text
10	Other

metrics. In addition, since this instrument has not been psychometrically validated, there is a possibility that there is unevenness in task difficulty that reduces the strength of these measures.

### 3.1.7 Data Collection

The instrument was a survey administered via Qualtrics. All participants had to consent to be in the study and then were asked demographic questions. Then Qualtrics automatically randomized them into four roughly balanced groups. Participants did not know they are assigned into a group nor that other groups existed. Each group was provided an explanation of what they need to know relative to the group they were assigned, and then they began their tasks. Midway through the assessment, participants were provided additional explanations relative to their assigned group to complete the rest of the tasks. The tasks consisted of a code sample, questions about the code, and questions about what helped and hindered them with arriving at the answer.

### 3.1.8 Quality of Measurements

As the study instrument was developed, participants were asked to provide feedback about their experience, which was used to improve subsequent versions. A question regarding the helpfulness or harmfulness of each visual attribute was repeated for each task. While we initially considered asking this question at one or two other areas of the instrument, asking after each task appears to have improved the quality of the measurements. This may be attributed to participants considering how they derived each answer immediately after answering rather than having to think back on how they assessed groups of questions over a longer period of time. It also likely helped account for differences in task goals and difficulty.

Figure 3.1: Colorful, In-line, Vertical Scroll

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange(1, 100)
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num, guess
  if guess > num
    return "high"
  elif guess < num
    return "low"
  else:
    return "correct"
while guess != num
  try:
    set guess = int(input("Choose an integer from 1 to 100: "))
  except:
    print(ERR_MSG)
    continue
  if guess not in guesses
    to list guesses append(guess)
  set result = check_num(num, guess)
  print("Your answer is", result)
print("It took", len(guesses), "guesses to guess correctly")
```

### 3.1.9 Instrumentation

There were a given set of tasks that all participants solved, but those tasks were presented with different visual coding cues for each group. The four groups as shown in table 3.1 are as follows: Group A received a colorful, vertically aligned rendering of a block-based program as seen in figure 3.1, Group B received a colorful, unaligned, horizontally dispersed rendering of a block-based program, Group C received a grayscale, vertically aligned rendering of a block-based program, and Group D received a grayscale, unaligned, horizontally dispersed rendering of a block-based program as seen in figure 3.2. The images were placed into an iframe to ensure each of the blocks in the layouts were the same size across all four groups and to force participants to scroll in their assigned direction. The full instrument is in appendix A).

The groups all received the same set of tasks, which included the code as described above, a sub-question about the code as described in table 3.3, and questions about what visual indicators helped or hindered them in deciphering the answer. The questions about visual indicators were the same for all sub-questions about the code. There were seven total code-related tasks, of which four were designated as locating tasks and three were interpreting tasks. All participants performed all tasks in the same order, so the only

Figure 3.2: Grayscale, Dispersed, Horizontal Scroll

```

set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange(1, 100)
set guess = 0
set guesses = []

def check_num(num, guess):
    if guess > num:
        return "high"
    elif guess < num:
        return "low"
    else:
        return "correct"

while guess != num:
    set guess = int(input("Choose an integer from 1 to 100: "))
    except print(ERR_MSG):
        continue
    if guess not in guesses:
        to list guesses append(guess)
    set result = check_num(num, guess)
    print("Your answer is", result)
    print("It took", len(guesses), "guesses to guess correctly")

```

Table 3.3: Group Attributes

#	Question	Designation	Type
1	How many times is a variable created in the code above?	Locating	Fill in the blank
2	How many times is a variable used (not set) in the code above?	Locating	Fill in the blank
3	How many strings are in the code above?	Locating	Fill in the blank
4	Which variable was used the most?	Locating	Mult. Choice (6 opts)
5	What is the first line of text displayed when the user runs the program?	Interpreting	Mult. Choice (12 opts)
6	Assume num is set to 95 and the user enters the integer 5. What is the next single line of text displayed to the user?	Interpreting	Mult. Choice (12 opts)
7	Assume num is set to 50 and the user enters "help" at the prompt. What is the next single line of text displayed to the user?	Interpreting	Mult. Choice (12 opts)

difference among groups were the code treatments as described above.

### 3.1.10 Masking

Participants neither knew that they were being randomly assigned to groups, nor did they know other groups existed. Since Qualtrics automated the randomization, no one administering nor assessing the study had influence over the group assignment. Participants self-reported their prior job experience, but this measure did not influence to which group they were assigned nor did participants know this metric would be analyzed or how.

### **3.1.11 Psychometrics**

Participants were asked to rate how much certain visual attributes helped or harmed them with each task. The rating was done on a 7-point Likert scale from a left-most position of “extremely harmful” to the right-most position of “extremely helpful” for each attribute. While there appears to be some disagreement in the psychology literature, Carmen et al. concluded that even though scale direction has some impact on responses, scale direction does not consistently affect response quality [142].

For this version of the survey, a slider mechanism was used for the Likert ratings. Funke et al. posited that radio buttons are better than sliders for break-off rates, anchoring effects, identifying intentional vs unintentional positioning, and higher response times [143]. While some of these, such as forcing participant positioning, can be mitigated, we should consider using radio buttons instead of sliders in future versions of this instrument. Beyond that, there were no psychometrically validated instruments ready to use for our research questions that we were aware of in the literature.

### **3.1.12 Random Assignment Method**

Since Qualtrics was utilized, randomization was done by creating a Qualtrics randomizer to set embedded data for each participant after the demographics data was presented. The embedded data consisted of a Group field that consisted of Group\_A, Group\_B, Group\_C, or Group\_D. The option to “evenly present elements” was selected to theoretically evenly distribute participants into groups. This randomizer did not take experience into account, which is why professionals were not evenly distributed into groups.

### **3.1.13 Random Assignment Implementation and Concealment**

Participants self-selected into the study and self-enrolled. The Qualtrics system automatically and randomly distributed them into groups roughly evenly.

### **3.1.14 Data Diagnostics**

Inclusion criteria were established to ensure participants took a minimum amount of time to consider the questions. The median time it took all participants to take the survey was 16.68 minutes with the first quartile finishing in 12.07 minutes. The base criterion was to spend at least ten minutes working on the tasks. If a subject took less than ten minutes, we hand-checked the data to inspect for obvious click throughs or someone injecting what could be fake data. If the data looked like it was faked, we removed

the participant. With this inspection, we found seven people that appeared to do this, leaving 87 in the study. This inclusion criterion was established because incentives were offered to take the survey, and some of them appeared to perform the fastest possible clicks to get to the end: e.g. rating all tasks the same on the scale or typing in what appeared to be junk in the text boxes. Qualtrics required participants to complete each task before moving on to the next task and every question had to be answered inside of each task. Abandoned and incomplete assessments were also excluded from the diagnostics.

### **3.1.15 Analytic Strategy**

The visual attribute data was subdivided and analyzed per attribute across all tasks in aggregate. We performed independent repeated measures ANOVAs on each attribute with the treatments of coloring, scrolling, and experience as interaction terms. When significance was found in the interactions, we performed post hoc tests to determine which interactions were significant. We also created box plots in the two-by-two-by-two layout to visually assess the ratings.

Timing and correctness were each considered in their own using repeated measures ANOVAs with the treatments of coloring, scrolling, and experience as interaction terms. When significance was found in the interactions, we performed post hoc tests to determine which interactions were significant.

## **3.2 Results**

### **3.2.1 Participant Flow**

Participants were recruited and joined the study over a 24-day period in June through July, 2023. The participants in each group are quantified in table 3.4 for a total of 87 participants across all groups. We discarded an additional 53 incomplete responses, most of which did not proceed further than the demographics or explanation sections. Only one made it to the beginning of the tasks, but they dropped out after the first question, which is why this participant was also discarded.

### **3.2.2 Recruitment**

Participants were recruited beginning two days before the first response through the first eight days of receiving responses. The responses for the rest of the period came in after the recruitment period ended. There was no follow-up after the initial recruitment period, as is typical under our Ethics board rules.

Table 3.4: Participants

Group	Classification	Per Classification	Group Total
A	Professional	14	25
	Novice	11	
B	Professional	4	19
	Novice	15	
C	Professional	3	19
	Novice	16	
D	Professional	9	24
	Novice	15	

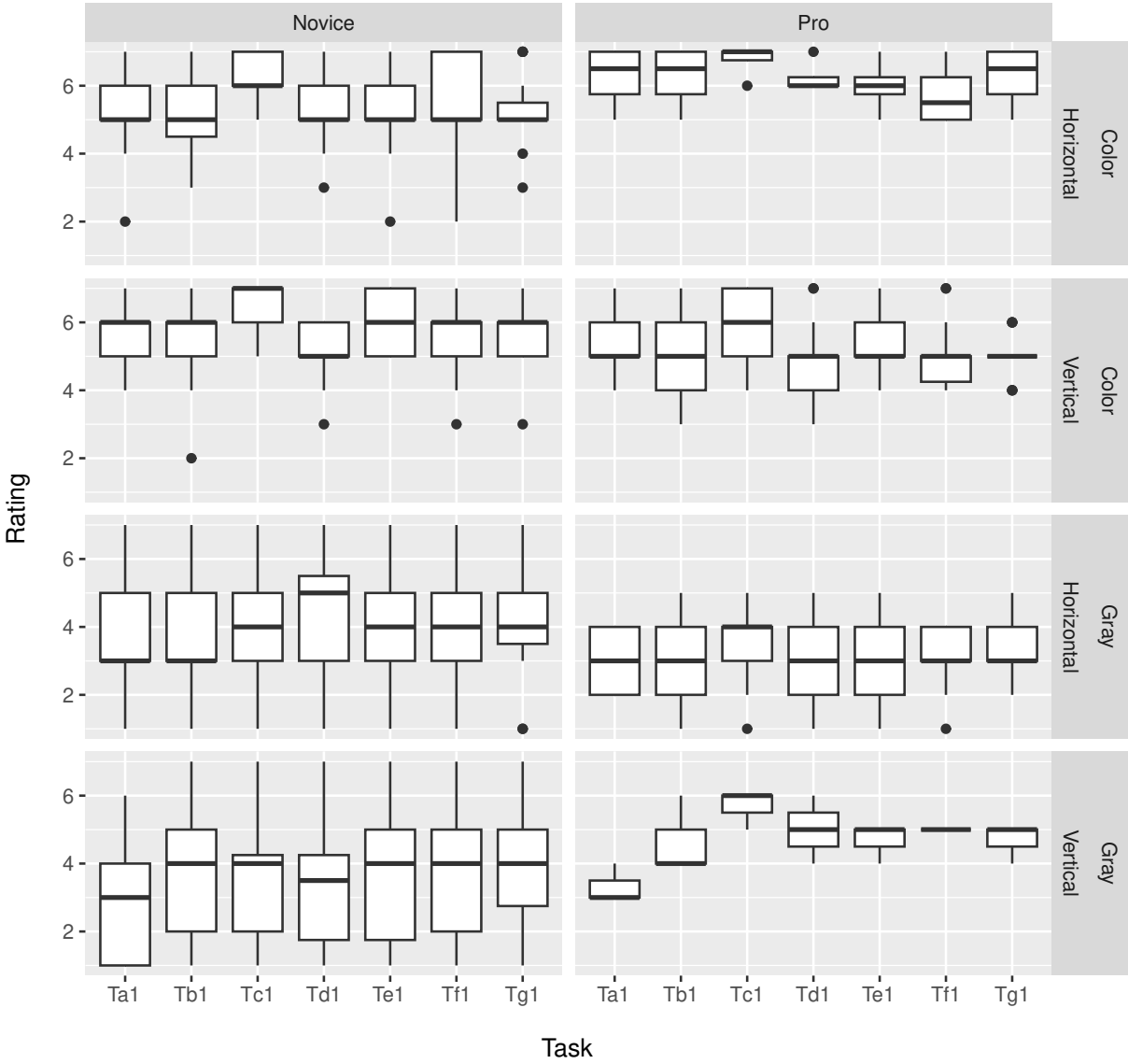
Follow-ups are allowed under the guidelines, but generally our ethical rule of thumb is to leave people alone if they do not want to participate.

### 3.2.3 Statistics and Data Analysis

We utilized a mixed factors repeated measures analysis of variance (RM-ANOVA) to analyze each attribute individually with a within-subjects predictor of task (to account for repeated measures) and between-subjects predictors of scroll direction, block coloring, and experience—with the requisite two-way and three-way interaction terms. This in turn leads to eight groups by dividing the original four treatment groups between the two types of experience: novices and professionals. The models estimate relationship within tasks and between predictor variables to arrive at the outcome variable, Likert rating. When running the RM-ANOVA, we chose to use Type II due to Langsrud’s [144] assertion that it is preferable to Type III. The following section is a per-attribute analysis for each of the models.

One of the fundamental assumptions in the univariate RM-ANOVA procedure is that of sphericity, which checks whether the variance/covariance matrix of the observed data from the Repeated Measures follows a particular pattern. If Mauchly’s test of sphericity indicates the assumption of sphericity is violated, we apply a Greenhouse-Geisser correction to the degrees of freedom used to calculate the F-ratio. These corrections often increase the p-value, so significance is reported again. Statistics were conducted and charts were created using RStudio Version 2023.09.0+463.

Figure 3.3: Block Color: Color Attribute Rating by Group and Experience  
 Color attribute rating by group and experience



## Color of the Blocks

The RM-ANOVA for this first attribute, “Color of the Blocks,” is statistically significant for the color treatment,  $F(1,79) = 44.94$ ,  $p < .001$ , generalized Eta squared ( $\eta^2G$ ) = .290, indicating that participants in the two groups who received the color treatment rated the block coloring differently than those in the grayscale treatment groups rated the color, or lack thereof.

Additional significance was found in the main effect of task  $F(6,474) = 5.61$ ,  $p < .001$ ,  $\eta^2G = .020$ , the color:task interaction  $F(6,474) = 4.17$ ,  $p < .001$ ,  $\eta^2G = .015$ , and the color:scroll:experience interaction  $F(1,79) = 7.63$ ,  $p = .007$ ,  $\eta^2G = .065$ .

The generalized Eta squared indicated the effect of the color treatment to be large at 29.0% of the model, ( $\eta^2G = .290$ ). The effect of the color:scroll:experience interaction is a nominal effect size of 6.5% of the model, ( $\eta^2G = .065$ ). The within task effects on their own and in interaction with color are quite small.

When analyzing Mauchly’s test, the data indicated that the sphericity assumption was not met for the color:task interaction ( $W = .59$ ,  $p < .001$ ), thus a Greenhouse-Geiser correction was applied ( $\epsilon = 0.86$ ). The interaction was still found to be significant ( $p < .001$ ) and generalized Eta squared indicated the effect when adding within-task differences was also small, ( $F(5.16, 407.64) = 4.17$ ,  $p < .001$ ,  $\eta^2G = .015$ ).

## Shape of the Blocks

The attribute labeled “Shape of the blocks” was significant for the color treatment,  $F(1,79) = 4.01$ ,  $p = .049$ ,  $\eta^2G = .027$ , indicating that participants in the two groups who received the color treatment rated the block shape differently than how those in the grayscale treatment groups rated the shape of the blocks. Task was also significant  $F(6,474) = 2.16$ ,  $p = .046$ ,  $\eta^2G = .012$ .

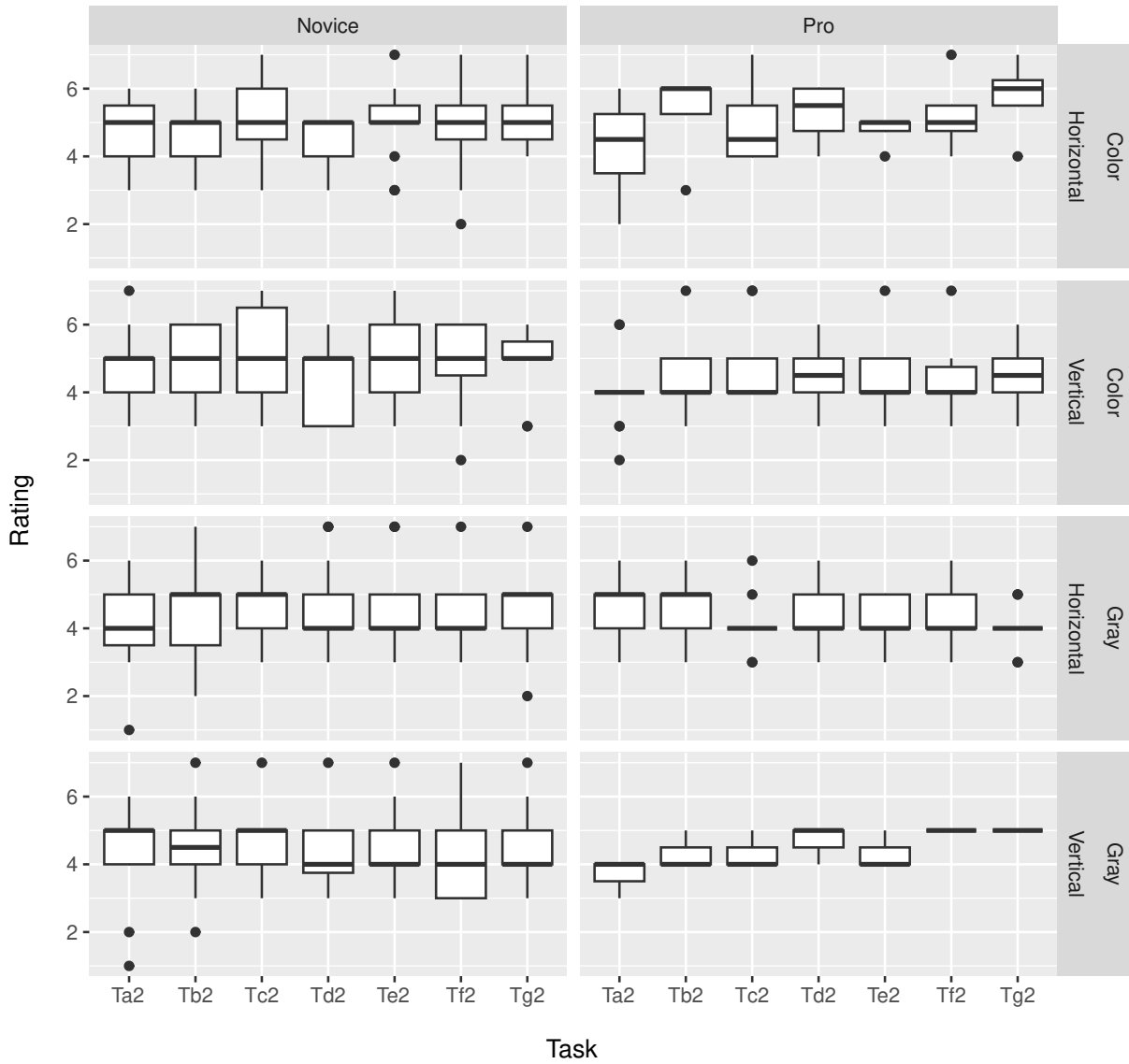
The generalized Eta squared indicated the effect of the color treatment to be small at 2.7% of the model, ( $\eta^2G = .027$ ). The effect of the task variable is a small portion of the model at 1.2%, ( $\eta^2G = .012$ ). None of the within task effects were significant. The boxplots per task and group are in figure 3.4.

## Arrangement of the Blocks

The “Arrangement of the Blocks” was significant for the color treatment,  $F(1,79) = 4.33$ ,  $p = .041$ ,  $\eta^2G = .029$ , indicating that participants in the two groups who received the color treatment rated the block arrangement differently than those in the grayscale treatment groups rated it. Additional significance was found in the interaction terms of color:scroll:task  $F(6,474) = 3.34$ ,  $p = .003$ ,  $\eta^2G = .019$ . The generalized



Figure 3.4: Shape of the Blocks: Block Shape Attribute Rating by Group and Experience  
 Block Shape attribute rating by group and experience



Eta squared indicated the effect of the color treatment is small at 2.9% of the model, ( $\eta^2G = .029$ ). The boxplots per task and group are in figure 3.5.

## Scrolling

The RM-ANOVA for the “Scrolling” attribute was significant for the scroll treatment,  $F(1,79) = 9.04$ ,  $p = .004$ ,  $\eta^2G = .067$ , indicating that participants in the two groups who received the vertical scroll treatment rated scrolling differently than those in the horizontal scroll treatment groups. The generalized Eta squared indicated the effect of the scroll treatment to be average at 6.7% of the model, ( $\eta^2G = .067$ ). The boxplots per group and task are in figure 3.6.

## Text, Punctuation, or Symbols

The attribute “Text, Punctuation, or Symbols” is statistically significant for the main effects and interaction effects for task,  $F(6,474) = 10.45$ ,  $p < .001$ ,  $\eta^2G = .048$ , the color:task interaction  $F(6,474) = 2.36$ ,  $p < .030$ ,  $\eta^2G = .011$ , and the scroll:experience:task interaction  $F(6,474) = 2.89$ ,  $p = .009$ ,  $\eta^2G = .013$ . This indicates that there were significant within task differences, which makes sense due to the nature that some tasks relied more on reading text than did others. The boxplots per task and group are in figure 3.7.

## Lack of Comments

When analyzing the attribute “Lack of Comments,” significant main effects and interaction effects were found for task,  $F(6,474) = 5.72$ ,  $p < .001$ ,  $\eta^2G = .015$ , and the scroll:task interaction  $F(6,474) = 2.58$ ,  $p = .018$ ,  $\eta^2G = .007$ . This indicates that there were significant within task differences for participants when rating the “Lack of Comments” attribute, which makes sense due to the nature that some tasks required interpreting code versus others that were just finding code elements such as variables. The boxplots per group and task are in figure 3.8.

## Lack of line numbers

The attribute “Lack of Line Numbers” displays significant main effects and interaction effects for task,  $F(6,474) = 4.76$ ,  $p < .001$ ,  $\eta^2G = .012$ , and the experience:task interaction  $F(6,474) = 2.27$ ,  $p = .004$ ,  $\eta^2G = .006$ . This indicates that there were significant within task differences for participants when rating

Figure 3.5: Arrangement of the Blocks: Block Arrangement Attribute Rating by Group and Experience  
 Block Arrangement attribute rating by group and experience

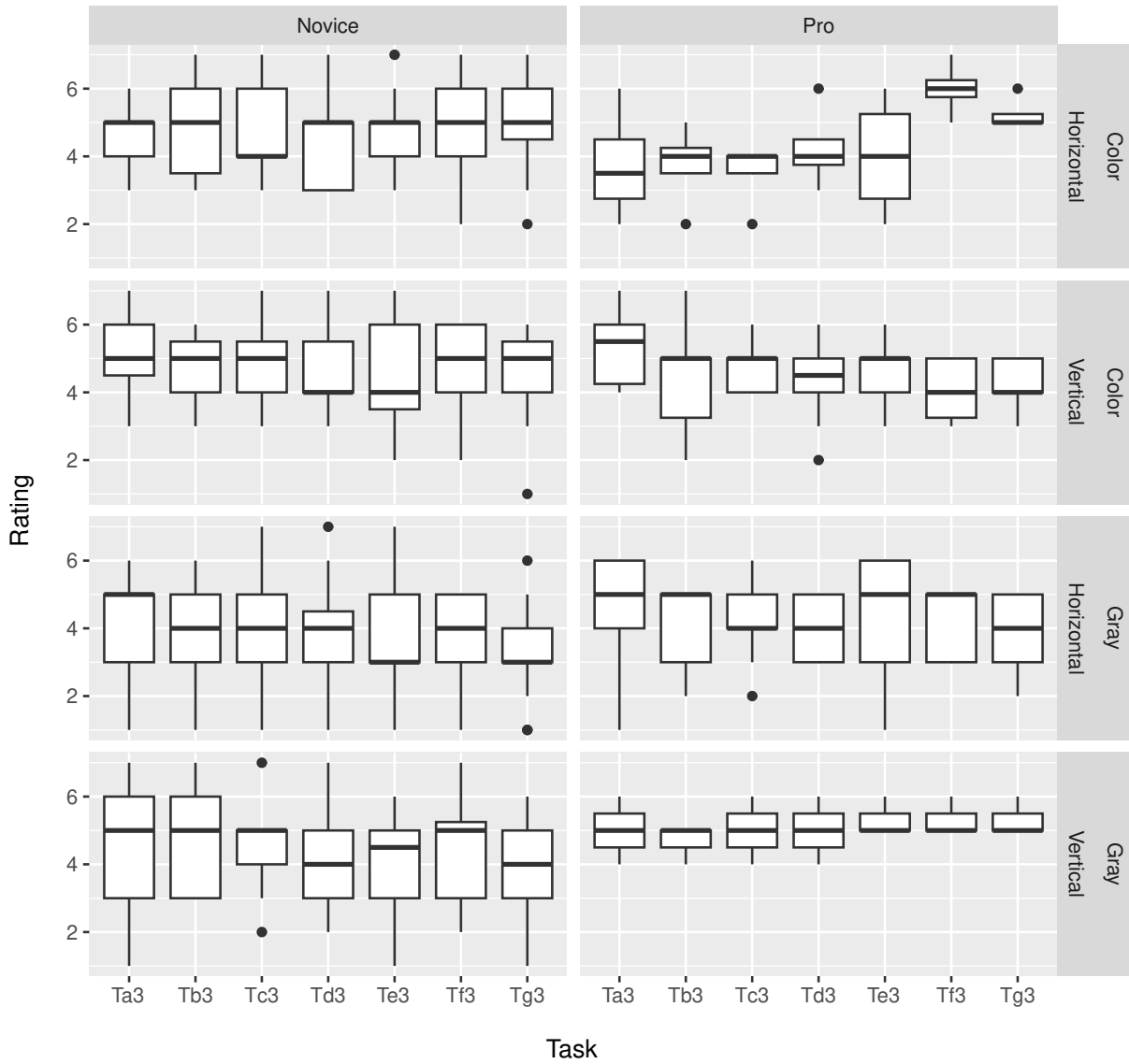


Figure 3.6: Scrolling Direction: Scroll Direction Attribute Rating by Group and Experience  
 Scroll Direction attribute rating by group and experience

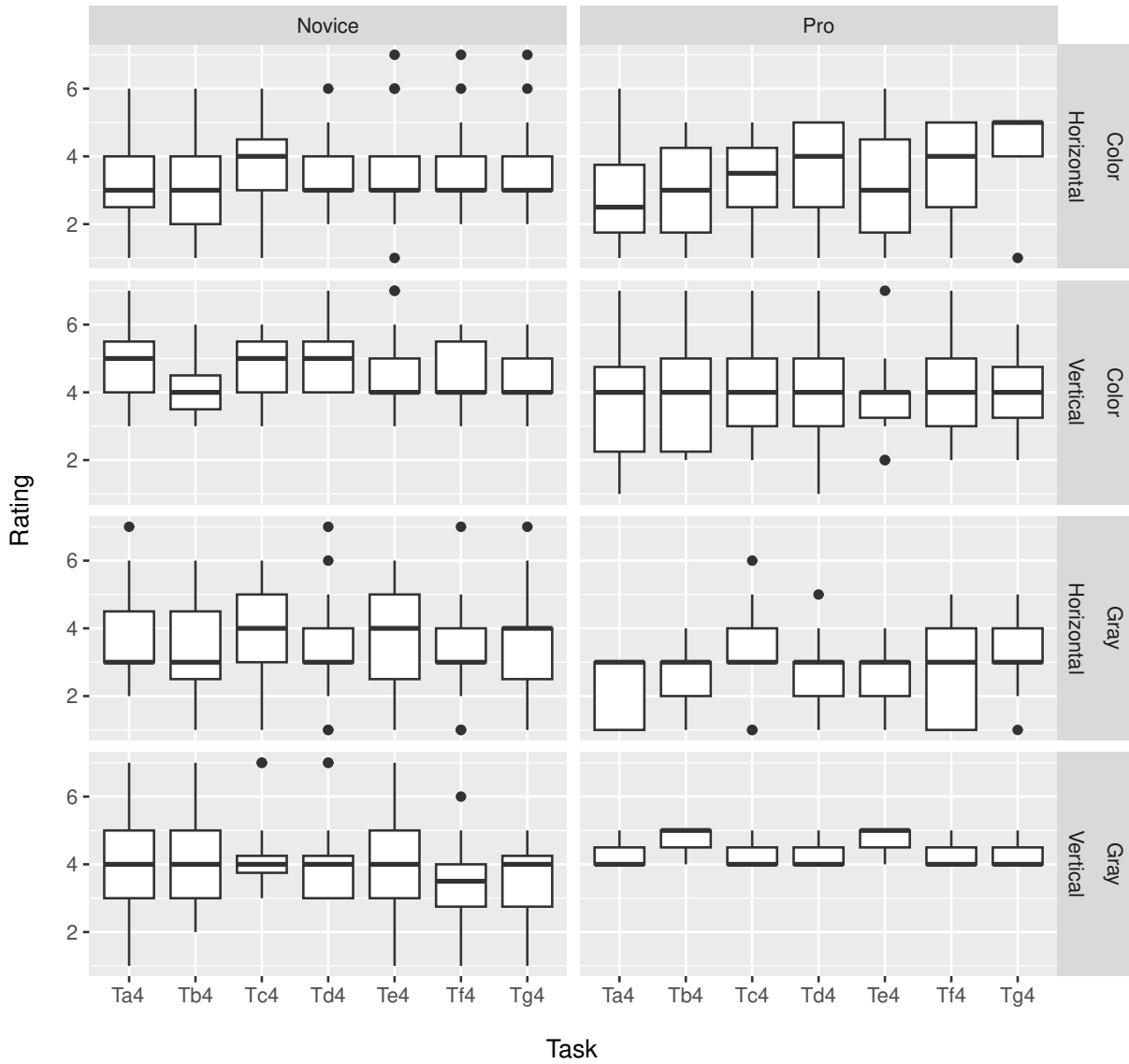


Figure 3.7: Text, Punctuation, & Symbols: Text, Punctuation, & Symbols Attribute Rating by Group and Experience

Text, Punctuation, & Symbols attribute rating by group and experience

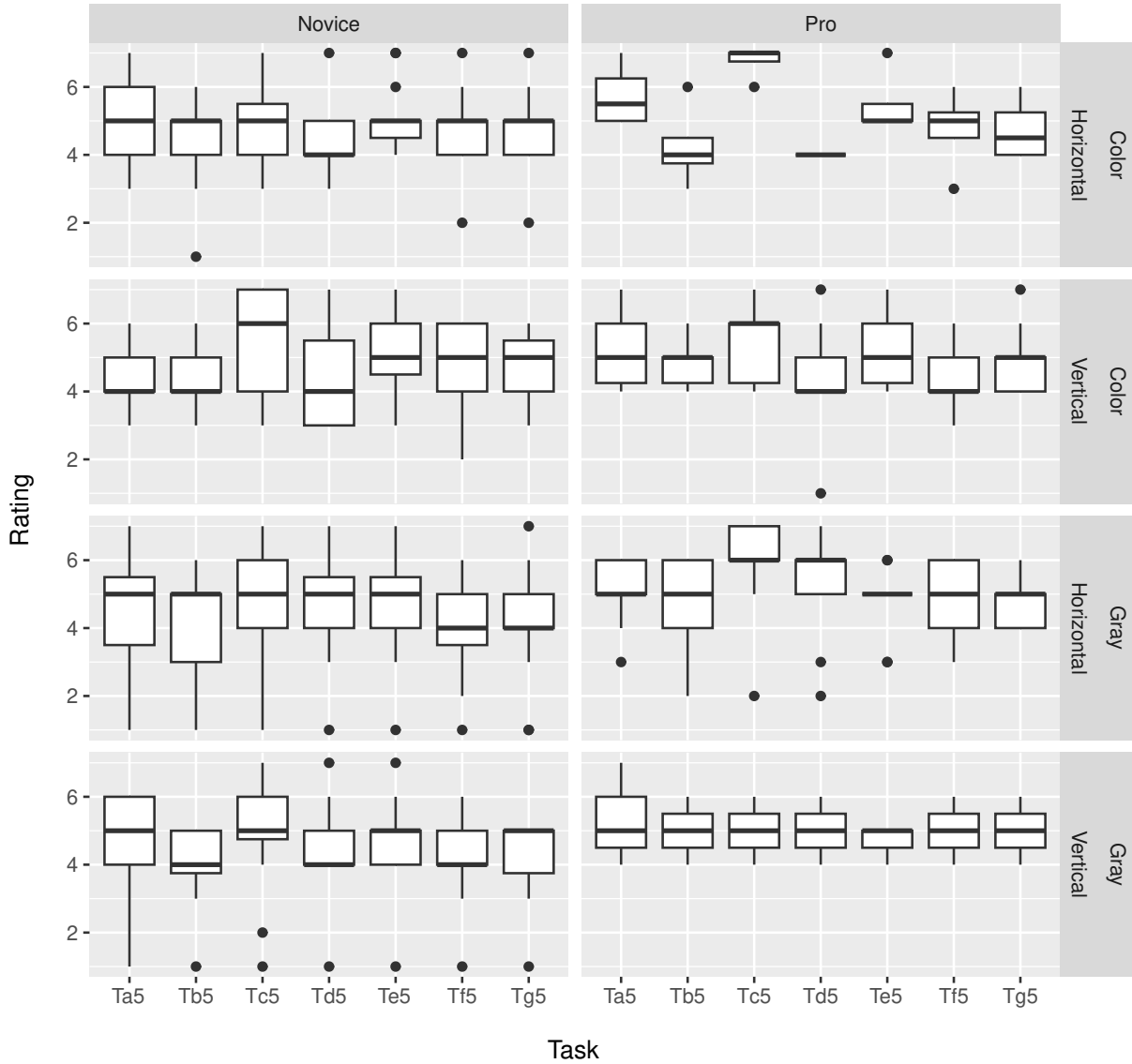
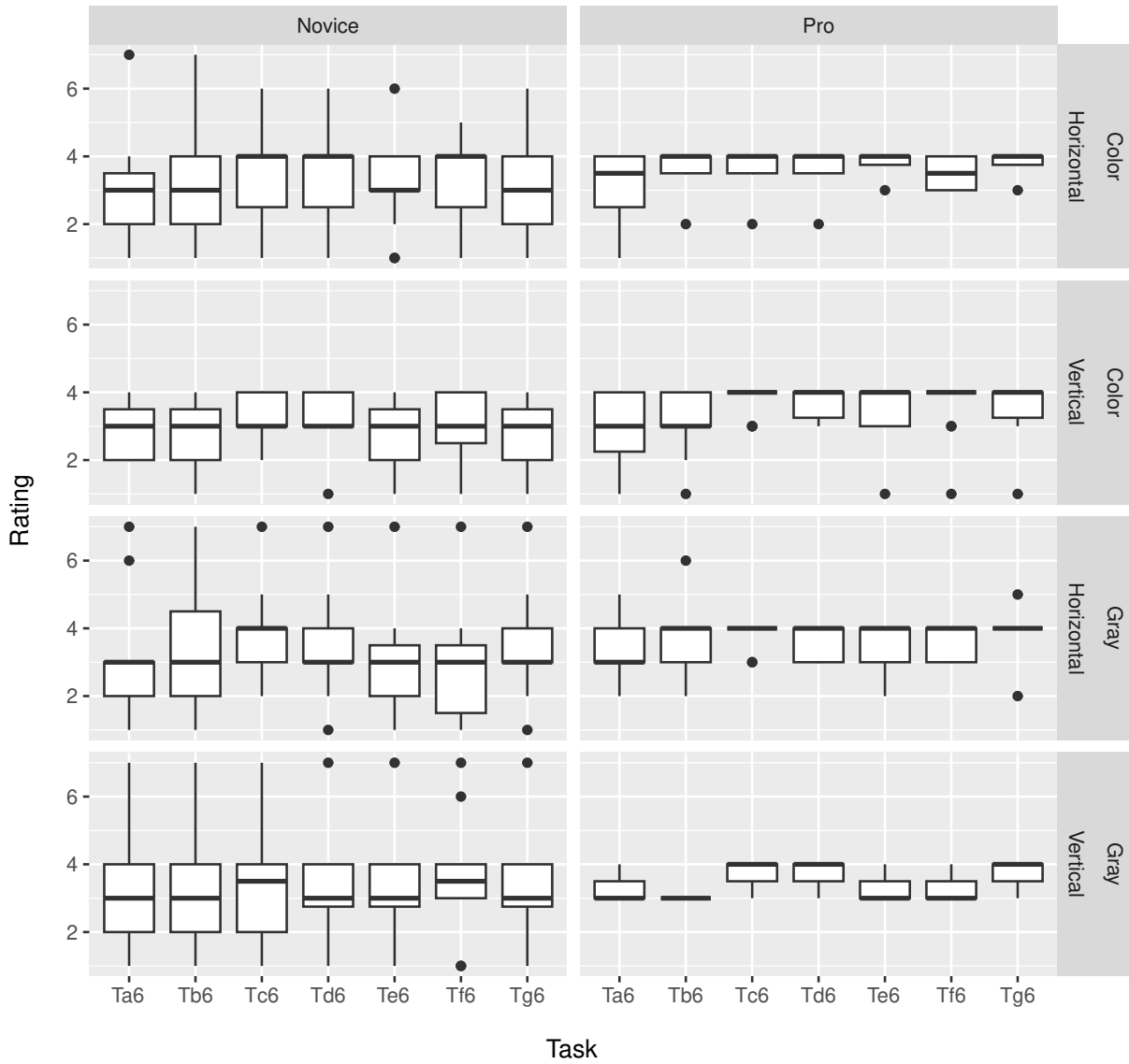
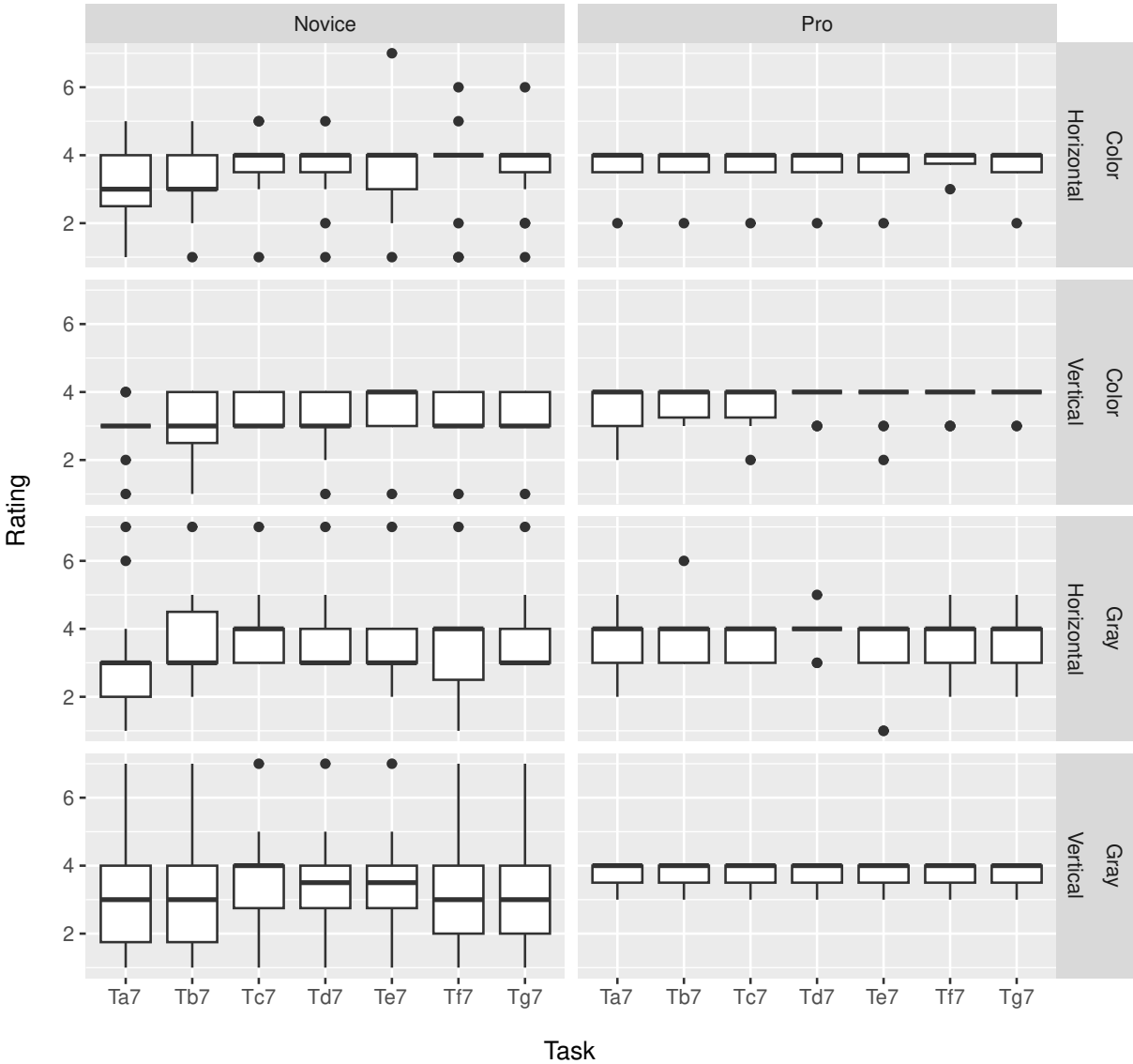


Figure 3.8: Lack of Comments: Lack of Comments Attribute Rating by Group and Experience  
 Lack of Comments attribute rating by group and experience



the “Lack of Line Numbers” attribute, which makes sense due to the variation in what participants were asked to do in each task. The boxplots per task and group are in figure 3.9.

Figure 3.9: Lack of Line Numbers: Lack of Line Numbers Attribute Rating by Group and Experience  
 Lack of Line Numbers attribute rating by group and experience



## Spacing

The RM-ANOVA for the “Spacing” attribute is statistically significant for the main effect of task,  $F(6,474) = 2.77$ ,  $p = .001$ ,  $\eta^2G = .012$ . When analyzing figure 3.10, it appears that most participants rated it generally as neither helpful nor harmful. The boxplots per group and task are in figure 3.10.

## Dropdown Arrow Next to Text

When analyzing the “Dropdown arrow next to the text,” which is an attribute of variables in this programming environment, we discover that there is significance in the main effect of the scroll treatment,  $F(1,79) = 5.05$ ,  $p = .003$ ,  $\eta^2G = .023$ , indicating that participants in the two groups who received the vertical scroll treatment rated usefulness of the dropdown arrow in the block differently than those in the horizontal scroll treatment groups rated it. Additional significance was found in the interaction terms of the scroll:task interaction  $F(6,474) = 3.63$ ,  $p = .002$ ,  $\eta^2G = .028$ . The generalized Eta squared indicated the effect of the scroll treatment is small at 2.3% of the model, ( $\eta^2G = .023$ ). The boxplots per task and group are in figure 3.11.

### 3.2.4 Timing and Correctness

When analyzing the task timing, we discovered there were 20 extreme outliers from 17 different participants, which reduces the quality of measurements, including half of the participant group comprised of professionals who received the color plus horizontal treatments. Additionally, professionals who received the color plus horizontal treatments comprised of only four participants and the group comprised of professionals who received the grayscale plus vertical treatments contained only three members. The numbers are reported here for both task timing and correctness with the caveat that these groups are too small to draw reliable conclusions.

The RM-ANOVA for task timing is statistically significant for task,  $F(6,474) = 13.56$ ,  $p < .001$ ,  $\eta^2G = .113$ , the experience:task interaction  $F(6,474) = 2.52$ ,  $p = .021$ ,  $\eta^2G = .023$ , the color:scroll:experience interaction  $F(1,79) = 7.72$ ,  $p = .007$ ,  $\eta^2G = .025$ , the color:scroll:task interaction  $F(6,474) = 2.74$ ,  $p = .012$ ,  $\eta^2G = .025$ , the color:scroll:experience:task interaction  $F(6,474) = 2.36$ ,  $p = .030$ ,  $\eta^2G = .021$ . The significance of the task main effect and each of the task interactions indicates that there were significant within task differences for the amount of time it took participants to complete each task. The primary interaction of note is the color:scroll:experience interaction, which indicates there is a significant difference



Figure 3.10: Spacing: Block Spacing Attribute Rating by Group and Experience  
 Block Spacing attribute rating by group and experience

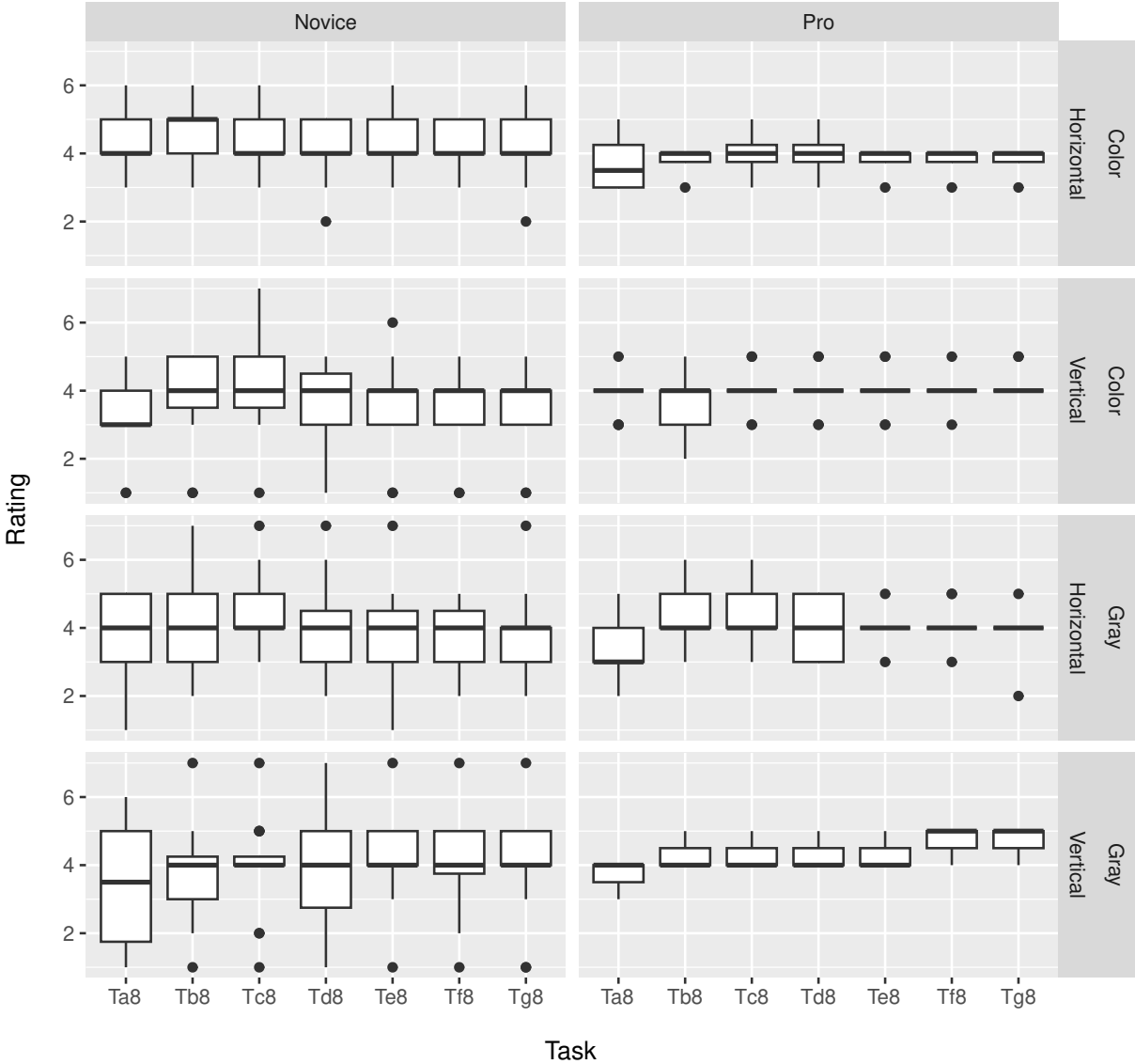


Figure 3.11: Dropdown Arrow: Dropdown Arrow Attribute Rating by Group and Experience  
 Dropdown Arrow attribute rating by group and experience

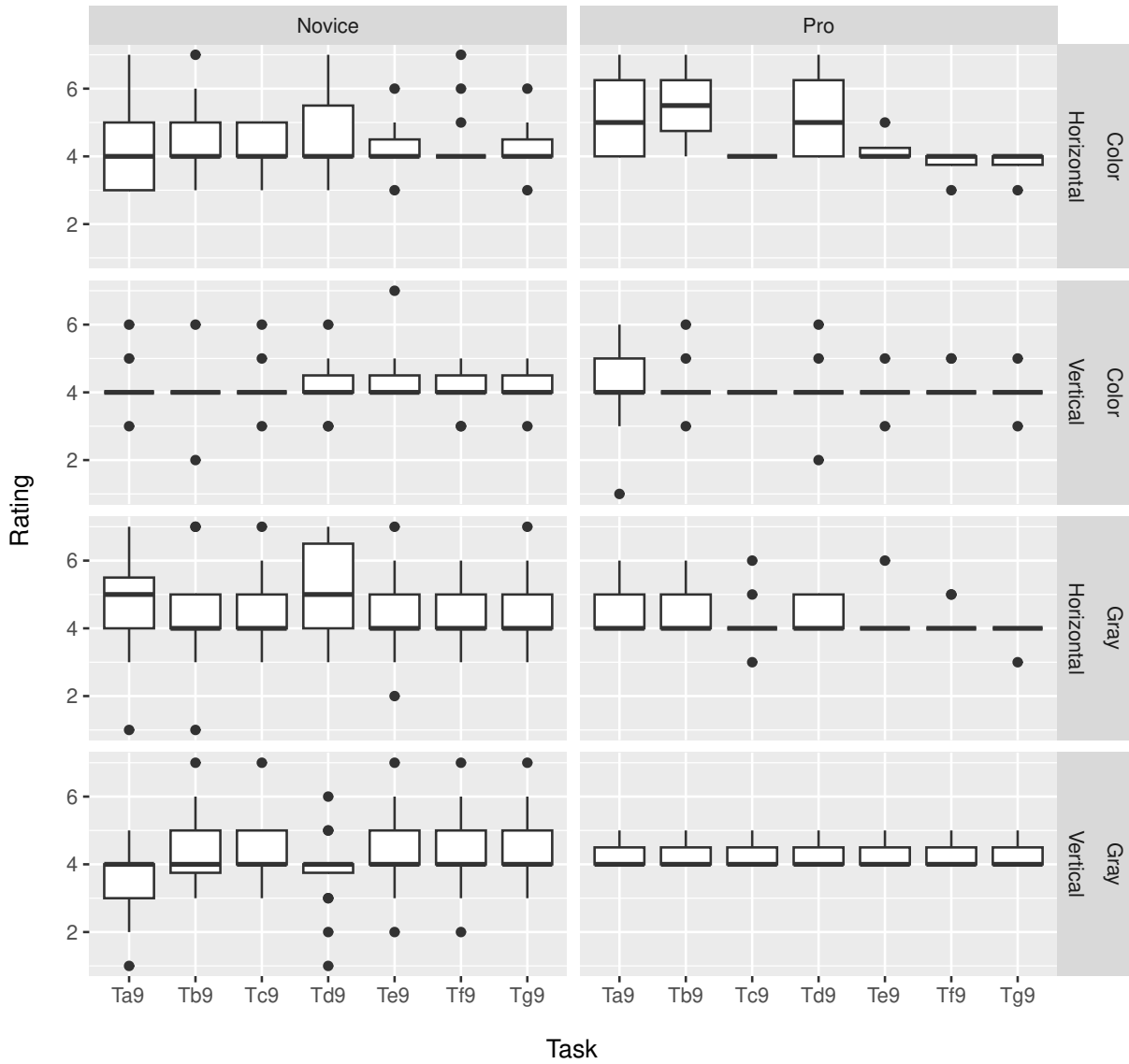


Table 3.5: Task Correctness Between Professionals and Novices

Experience	Mean	SD
Novice	0.484	0.500
Professional	0.595	0.492

with this three-way interaction.

The RM-ANOVA for task correctness is statistically significant for the main effects experience,  $F(1,79) = 6.31$ ,  $p = .014$ ,  $\eta^2G = .020$  and task  $F(6,474) = 50.96$ ,  $p < .001$ ,  $\eta^2G = .323$ . As expected, the greatest strength of the model is based on task as 32.3% whereas experience explains 2.0% of the model. Table 3.5 shows the mean and standard deviation of correctness for professionals and novices on a scale of zero to one.

### 3.3 Discussion

#### 3.3.1 Support of Original Hypotheses

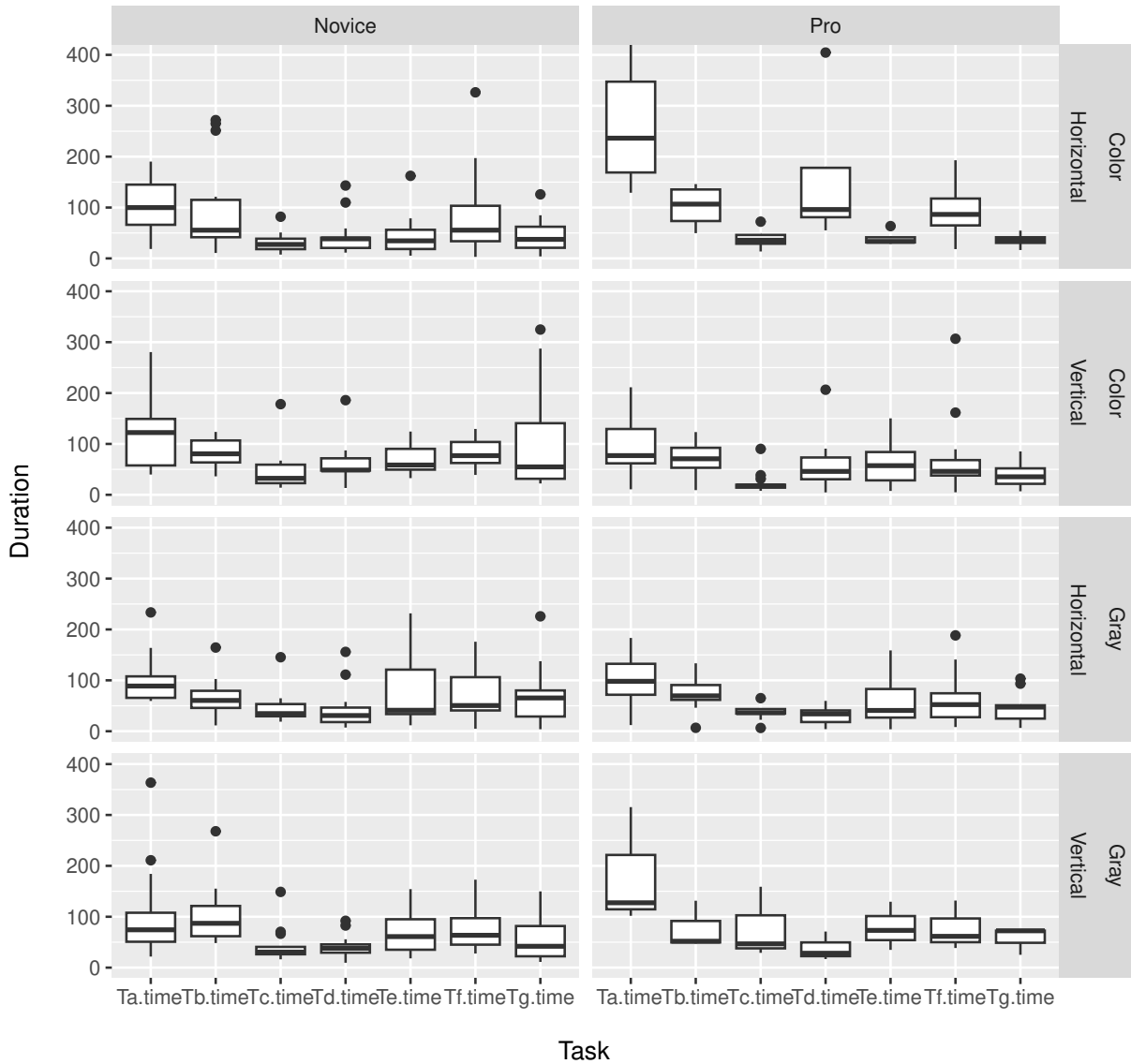
Due to the limited participant pool, we were unable to find statistically significant differences between novices and professionals, so a future study should be conducted with a larger group. Regardless, we were able to gain some valuable insight into what visual attributes are helpful and harmful. In order to test for significance of correctness, we would need a much larger participant pool. With that, we hope to see if those hypothesis bear out with correctness data added. Until then, research questions one and two can at a maximum be partially supported.

#### 3.3.2 Hypothesis 1

*H<sub>0</sub>: There will be no difference in preference or correctness between groups given the block colors or grayscale treatments.*

Block colors were preferred as a means to help participants when answering questions, thus we partially reject the null hypothesis 1 in favor of the alternate hypothesis: there is a difference in preference between the groups given the block colors versus the grayscale treatment. This finding was supported for both novices and professionals. When analyzing at the post hoc comparisons, we see that novices rated color and grayscale differently within the same scroll groups, but professionals only rated a significant difference in color versus grayscale in the horizontal scroll group. The groups that received the color treatment rated

Figure 3.12: Task Timing by Group: Duration per Task by Group and Experience  
 Duration per task by group and experience



color of the blocks as beneficial, whereas those with the grayscale treatment rated the color of the blocks to be detrimental. Surprisingly, professionals in the vertical groups were neutral to colorful blocks versus grayscale blocks. Additionally, there was a significant difference between how professionals and novices rated the attribute between scroll directions when the blocks were in grayscale. Professionals rated the color attribute much lower in the gray-horizontal treatment group than did novices, and professionals rated the color attribute much higher in the color-vertical treatment group than novices did. This could be due to various factors such as novices' lack of experience, varying degrees of experience within the novices group, or the limited pool of participants.

### 3.3.3 Hypothesis 2

*H<sub>0</sub>: There will be no difference in preference or correctness between groups given the single-aligned vertical block layout arrangement or the spatial (dispersed, horizontal) layout.*

Single-aligned block layout with vertical scrolling was preferred over dispersed, horizontally scrolling layouts, thus we partially reject the null hypothesis 2 in favor of the alternate hypothesis: there is a difference in preference between the vertical vs spatial layouts. For both professionals and novices, vertical scrolling was rated significantly higher than horizontal scrolling. In both horizontal-scrolling treatment groups, horizontal scrolling was seen as very harmful to completing the task at hand. Professionals found horizontal scrolling to be much more harmful than did novices, which again could be due in part that it is different from their accustomed layout and practices.

In the case of vertical scrolling, professionals were somewhat neutral to the helpfulness or harmfulness, likely because this is what they are used to doing, thus scrolling vertically was seen as normal rather than specifically beneficial. Novices found scrolling vertically to be somewhat helpful, which is likely a matter of rating the capability and necessity of scrolling itself, rather than the direction of it, as slightly positive. The slightly positive rating from novices might be attributed to the good subject effect, but that is difficult to fully ascertain with the given data.

### 3.3.4 Hypothesis 3

*H<sub>0</sub>: There will be no difference in preference for block shapes regardless of treatment group or experience.*

There was a difference in the preference for block shapes between the color and grayscale groups, thus we reject the null hypothesis 3 in favor of the alternate hypothesis: there is a difference in preference between

treatment groups. When analyzing figure 3.4 we can see this is especially true for professionals, who found the shape of the block to be very helpful in the color-horizontal treatment group yet were more neutral in the grayscale vertical treatment group. Professionals tended to rate the shape of the blocks as less helpful in each of the grayscale treatment groups relative to the groups that received the color treatment. This indicates that professionals use visual cues differently, and perhaps in a more refined manner, than novices who are still developing their programming skills. These findings should be considered further for color blind and visually-impaired individuals, especially when horizontal layout is allowed or enforced by the language design, since block shape could help compensate for visual impairments.

### 3.3.5 Hypothesis 4

*H<sub>0</sub>: There will be no preference for the lack of comments and line numbers.*

Since all groups received the same treatment without comments and without line numbers, we were unable to truly test for significance of these variables. Thus we fail to reject the null hypothesis. Using this data, we can look at the average Likert ratings for “Lack of comments” and “Lack of line numbers” to help design a future study. Overall, the lack of comments was rated as harmful,  $M = 3.30$ ,  $SD = 1.19$ . Novices found the lack of comments more harmful,  $M = 3.18$ ,  $SD = 1.35$ , when they performed tasks than professionals did,  $M = 3.54$ ,  $SD = 0.76$ . Likewise, the lack of line numbers was also reported as harmful overall,  $M = 3.47$ ,  $SD = 1.11$ . Again, novices found the lack of line numbers to be more harmful,  $M = 3.37$ ,  $SD = 1.27$ , when completing tasks than professionals did,  $M = 3.66$ ,  $SD = 0.70$ .

### 3.3.6 Similarity of Results

Block colors and color categorization has been a point of discussion in the literature. Weintrop et al. posited, via positive anecdotal responses, that professionals appreciate block colors as they can see how the blocks convey syntactic information [19]. This work provides quantitative support for such a conclusion.

### 3.3.7 Interpretation

This study aimed to quantify what visual attributes were determined to be helpful or harmful for completing tasks correctly and quickly. With the limited participant pool, we were not able to test support for correctness, but we were able to gain some valuable insight into what visual attributes are helpful and harmful. In order to test for significance of correctness, we would need a much larger participant pool.

Colors and color categorization are beneficial for both novices and professionals. While there is still debate as to which colors to use or not use [145, 146], as well as how large the colorful block should be, color is beneficial for people without visual impairment and is critical to assess for those with visual impairment. Determining how to use colors and categorize colors would be a beneficial step forward for a future work. Any future work should also take into account people with colorblindness and other visual impairments.

The results are at least suggestive that block-based programming environments should rethink the open canvas idea in favor of block alignment to a left-margin (for left-to-right languages). Predominantly horizontal scrolling was viewed negatively by novices and professionals alike. Programming environments with a forced horizontal scroll, such as Lego Mindstorms [147] and Scratch Jr. [148], should study this finding further. Their results may be different because they work with the very young, but so far as we can tell from the literature, if this is effective for such students, the anecdotal claims about it in the literature are not terribly convincing. If anything, allowing blocks to be randomly placed on the palette and still be executed with the rest of the code seems detrimental and should be further tested. Keeping blocks aligned to a side margin and maintaining a vertical, evenly spaced alignment may also have transfer of learning impacts.

Distinctive block shapes were rated positively to task completion across the board for novices, but it was split for professionals. While we found no impact for correctness, the benefits of block shapes were circumspect when testing it against color versus grayscale for professionals. This attribute needs to be studied further. Overall, the “text, punctuation, and symbols” attribute was rated as helpful across all groups,  $M = 4.74$ ,  $SD = 1.27$ , which indicates that block shapes do not replace the need for carefully considering the verbiage on the blocks.

The lack of comments and lack of line numbers were viewed as detrimental by novices and professionals alike. While blocks may seem more simple to read and understand at face value, having the abilities to comment and reference lines are generally considered to be beneficial and to be best practices for programming. We want to be clear that novice and professional opinion is just that and nothing more. Thus the evidence here does suggest that this was the opinion in the context of our sample, but this is not sufficient to guarantee generalizability.

### **3.3.8 Generalizability**

This study encompassed participants ranging from novices to professionals. All participants completed high-school at a minimum, up through some of whom had a PhD. Professionals were those who had in the

past or currently are employed in jobs where programming is at least a partial, if not total, aspect of their professional work. Despite these factors, we cannot claim broad generalizability to adults of varying skill levels. Additionally, it cannot generalize K–12 students due to the exclusion of that group; thus future replications should include this population to discover similarities and differences in this group.

Further, studies involving surveys have natural limitations. First, we are unaware of any study in the literature that has psychometrically validated the instrument used, including ours. Doing so is a long process, in part because one could reasonably object to almost any wording chosen in the study, understandably. Past that, exactly which method to use for validation here is not entirely clear. For example, typical factor analysis questions correlation because there is an underlying assumption that a measurement might be measuring a latent variable. Here, however, there is no underlying hypothesis of a latent variable because each question is intended to draw out whether or not a particular design constraint exists. For example, line numbers questions “could” correlate with questions about shape, but just gathering a factor loading misses the point. The lack of psychometric validation is another cause for push back against potential generalizability.

In a future work, one could hypothetically build a block language with one of these features in, or out, and then correlate some external metrics to establish whether or not these questions correlate with real-world performance (e.g., grades in a class, performance in a job). As our goal here was to establish this survey to put limiting parameters around the scope of potential options to study, we leave such questions to future work.

### **3.3.9 Implications**

Future research should be conducted in various ways. First and foremost, this study could be conducted on a larger sample, which would afford the ability to test the metrics against a correctness metric. It is of limited use when participants rate an attribute as extremely helpful only to discover they had the wrong answer. Additionally, it would be beneficial to determine timing metrics to determine if attributes rated as helpful or harmful contributed to solving problems faster or taking longer, respectively. We suspect that moving to a more sophisticated testing environment, rather than testing within Qualtrics, would help in this regard. Finally, testing this instrument on K–12 novices would increase the generalizability of these findings. Ideally, it makes sense to consider “birth to death” as a metaphor, if we really want to understand when, and where, such results do or do not apply.



## Chapter 4

# Study 2: Investigating a Context-Aware Palette for a Block-Based Language

### 4.1 Background

To create block-based programs, learners and programmers need a code editor to write programs and a way to run them. Due to the graphical nature of block languages, an integrated development environment (IDE) is a helpful, if not required, resource. IDEs are commonly used by programmers to perform their primary function of developing code in any type of programming languages, so this study leverages the experience participants gained from using programming IDEs. IDEs typically consist of, at a minimum, a source code editor, build and run capabilities, and a debugger all bundled into one interface. They also often contain other capabilities, such as a source code revision control, class explorer, static code analysis, and code completion as highlighted by Tran [149], to help programmers write code more quickly and effectively. Lin et al. [140] found that block-based IDEs have up to 24 categories of blocks, some of which employ nesting categories in an attempt to not overwhelm users with too many categories. None were cited as having a context-aware palette of blocks, nor did they state what additional capabilities they had, if any.

These capabilities are called affordances, in Gaver's [150] terminology, which was adapted from Norman [151], on human computer interaction (HCI). In the physical world, a vertical door handle implies the affordance of pulling it open, whereas a vertical bar on an exit door implies the affordance of pushing it open. Perceptible affordances offer a direct link between a person's perception and the action they take, but hidden affordances and false affordances limit or hinder a person's ability to effectively use the soft-

ware. For a context-aware palette in an IDE, a perceptible affordance would offer information about what actions could be taken with each element of the palette, such as blocks being dragged from the palette into the editor. Affordances can be auditory in nature as well, such as a defined noise for errors, which could improve accessibility. In Gaver’s treatise on technology affordances in HCI, the suggestion is to focus on the interaction between technologies and users to assess the strengths and weaknesses of the affordance in order to design software that will assist users.

Xia et al. [152] found that developers spent 24.8% of time in navigation tasks (using the IDE, finding code, etc.) and only 6.4% of their time actually writing code. Since developers spend nearly four times the amount of time working *with* the IDE rather than writing code *in* the IDE, having the right tools, features, and capabilities may be of benefit according to the available data. Additionally, as discussed previously, Dwyer et al. [87] concluded that their block-based IDE itself created sources of information and misinformation for users; thus, they suggested researchers should consider what visuals and capabilities should be provided in a block-based IDE.

## 4.2 Methods

### 4.2.1 Hypotheses

In pursuit of assessing developers’ perceptions of IDE tools that are beneficial, we formulated the following null hypotheses:

1. Professionals and novices will consider all tools to provide the same helpfulness.
2. Professionals and novices will want the same tools regardless of the scenario.
3. Professionals and novices will have the same viewpoint on the helpfulness of the tools.

### 4.2.2 Inclusion and Exclusion

Recruitment to this study was limited to adults either actively enrolled in a computer science major or minor at the University of Nevada, Las Vegas or adults who are or were professional programmers. Potential participants had the option to consent to be in the study after they opened the link to the study. If they consented to be in the study, demographic questions were asked initially to ascertain each participant’s educational attainment, current level if still in college, any professional history writing code, and which programming languages were previously used to write code. The demographic questions did not have any

Table 4.1: Participants

Group	Classification	Per Classification	Group Total
Professional	Not Enrolled	7	21
	Graduate Program	2	
	Senior	6	
	Junior	5	
	Sophomore	1	
Late Stage	Graduate Program	1	76
Novice	Senior	38	
	Junior	37	
Early Stage	Sophomore	14	26
Novice	First-Year	12	

influence on participant inclusion or exclusion from the study. These data were collected solely for the purpose of determining how each participant’s level of experience (early-stage learner, late-stage learner, professional) affected their responses. The only data included in the study was from participants who consented to be in the study, answered all of the demographics questions, and answered all of the study questions. In any other case, the participant data was excluded from the study.

### 4.2.3 Participant Characteristics

As in the first study, both professional programmers and university students were recruited in order to analyze potential differences in responses. Professional programmers were solicited via direct email while University students were recruited from the authors’ institution’s undergraduate-level computer science program. If students had professional experience listed as “Never employed to write code as a primary job function, but it was a periodic part of the job”, they were included in the late-stage learner category regardless of their classified year. If students chose any other type of professional experience, they were categorized as professionals.

### 4.2.4 Sampling Procedures

Participants were solicited via email across seven different companies. Additionally, participants were recruited from four undergraduate-level university courses at the authors’ institution. Solicited participants then had the option to self-select into the study. There were a total of 123 participants, 21 of which were classified as professionals and 102 of which were classified as learners. Early-stage learners, first-years, and sophomores comprised of 26 of the learners and late-stage learners, juniors, and seniors formed the

remaining 76 learners. Table 4.1 shows a full breakdown. This classification was created from the self-reported responses to the demographics questions. One question asked if the individual was currently enrolled in a degree program, and if so at what level, and another question asked if they were then currently or had ever previously been employed to write code as all or part of their job function. If a participant conformed to both categories, they were categorized in the professional group. All participants, regardless of categorization, performed the same tasks in the same manner for this study.

#### 4.2.5 Sample Size, Power, and Precision

Similar to the previous study, we are unaware of an existing similar study, so there was no direct way to calculate our required sample size, power, or precision. Our approach was to undertake a series of pilot studies to provide participant feedback on the study. None of the participants in these pilots were included nor reported in this study. For the first pilot, we started with a small sample size, then roughly doubled from one major pilot version to the next. While the pilot data is consistent with what is reported here, it is not directly comparable due to changes in questions and experimental design. We highlight it in this section because it was an important part of our process in creating the final instrument.

#### 4.2.6 Measures and Covariates

The instrument displayed a snippet of code in the block-based Quorum programming language along with a prompt of the goal that the in-progress code snippet was intended to perform. The participant was asked to imagine they were completing the goal of the given programming task along with being shown a green arrow of where their cursor was active in the code.

Participants were given 16 potential types of tools, which could alternatively be called features, functionalities, or affordances, that could be displayed to help them with completing the programming challenge. Participants were asked to categorize each of these tools as either “Helpful” or “Not Helpful.” They were then required to rank the tools they had categorized as “Helpful,” with “1” being the “most helpful” tool in each given context.

Since the tasks were unrelated, and their context was critical for the purpose of this study on a context-aware IDE, each task was statistically assessed individually. The 16 types of tools are presented in table 4.2 in the order in which they appear in the instrument.

Table 4.2: IDE Potential Types of Tools

#	Tool Type
1	Project Files/Structure
2	Source Control
3	Class Name/Library
4	Class Short Description
5	Blocks of In-Scope Variables
6	Alert About What Problems
7	Link to Class Documentation
8	Blocks to Create New Variables
9	Blocks for Control
10	Blocks for Conditionals
11	Blocks of All Functions in the Class
12	Blocks for I/O
13	Blocks to Create New Blocks
14	Blocks for Exiting
15	Blocks to Import Additional Libraries
16	Example Code

#### 4.2.7 Data Collection

Qualtrics was again utilized to administer the study instrument. All participants were presented the same set of questions and tasks. Participants were required to consent to participate in the study, and were then asked demographic questions and presented an explanation of the purpose and particulars of the study. They were then asked a series of questions: one starting with a free-form inquiry about what the IDE could suggest to help them, six questions about sorting and ranking predefined helper-tools, and then a final free-form question about what the IDE could suggest to mirror the first question.

#### 4.2.8 Quality of Measurements

Study development consisted initially of creating multiple different study design ideas to the research group, where we went through multiple iterations with review. We then solicited experiential pilot-participant feedback with modifications to improve subsequent study prototypes. While we initially considered showing various potential renderings of suggestions, we determined that there were two main issues with this approach. First, the number of possible options for display could divide participants into too many groups to perform a feasible study. Second, pilot feedback indicated that the rendering anchored participants to a certain viewpoint, which seemed to taint the results. Iterative pilot study results led us to present participants with a programming prompt along with a list of potential suggestions to sort and rank into

Figure 4.1: Categorization and Rank Questions

#	Question	Line of Code	Where Arrow Points
1	Imagine you've been tasked to write a small function that will take in an integer array, prints out the even integers, and then returns an array of the even integers to the caller.	<code>use Libraries.Containers.Array</code>	Anywhere on the line
2	Same as above	<code>action PrintEven(Array&lt;integer&gt; array)</code>	Anywhere on the line
3	Same as above	<code>if</code>	At the end of the line to complete the if statement
4	Imagine you've been tasked to write a small function that takes in two integers and adds them together.	<code>Add(</code>	At the end of the line to complete the function call
5	Imagine you're doing some statistical analysis, and need to load data into a dataframe.	<code>DataFrame frame</code>	Anywhere on the line
6	Imagine you're doing some statistical analysis, and you need to display the chart or make some edits to the chart.	<code>chart:</code>	At the end of the line to complete the object reference

helpfulness categories. While we went through several small scale pilots to arrive at this instrument, this is the first time the full tool has been conducted at scale.

#### 4.2.9 Instrumentation

The ad hoc instrument consisted of eight questions: two were free-form text and six consisted of categorizing and ranking (full instrument is in appendix B). The instrument first broadly contextualized the study, and then asked the first free-form text question. Next came the six categorization and ranking questions. Finally, the participants were asked to input a free-form text answer to a question that mirrored the first question. All study participants received the same questions in the same order. The six categorization and rank questions are shown in table 4.1.

#### 4.2.10 Masking

Participants did not know the study design nor the fact that there was only a single group either prior to nor during the study. Participants self-reported their status as student or non-student, any relevant student classification, and prior job experience, but they did not know if or how these metrics would be analyzed.

#### 4.2.11 Psychometrics

First, we are unaware of any study using a psychometrically-validated instrument to examine context-aware integrated development environments (IDEs), including ours. Psychometrically validating an instrument is a lengthy process, for the reasons stated previously. We initially sought out similar studies that have been psychometrically validated by task-technology fit theory (TTF). To identify relevant studies, a search was conducted in the Scopus, IEEE, and ACM research databases. They were chosen due to the large number of journals and conferences covering a wide array of disciplines, and they are fundamental to computer science research. To be considered a relevant study, it had to be published in a peer-reviewed journal or conference article applying TTF theory and resulting in a psychometrically-validated instrument. No such instruments nor close analogues were found.

This cross-sectional observational study was created to gather data on how professionals and novices think about the importance of IDE tools across different scenarios when programming. While the study asked participants to think about what tools they might need in a given scenario, the participants were not asked to write any code to implement the given scenario. They were only given the scenario, a snippet of code, and told where the cursor was placed to provide a frame for their thoughts. All participants received the same scenarios in the same order regardless of their answers to the demographics questions.

Participants were asked to categorize types of tools that an IDE could display in the left-hand pane. They were asked to categorize each tool as either “Helpful” or “Not Helpful” in the given context of each question. Participants were then asked to rank the items they placed into the “Helpful” category with the first ranking being the most helpful in that context. While we did not initially know which tools would be ranked as most important, through the piloting process, we were able to narrow down the tool list to the 16 tools we chose to be in the study. However, it is possible that some important tools were missing from the study. Since this is an initial study on the subject, our aim was to develop a better picture of the initial tools, or affordances, that would provide the most assistance to programmers in the initial version.

As our goal was to establish this survey for such a purpose, we leave psychometric validation and TTF theory about IDEs to future work.

#### **4.2.12 Participant Selection**

The goal with participant selection was to ensure we had representation from various experience groups. While the participation rate per group was not evenly distributed, as can be seen in table 4.1, we were able to recruit participants from first-year of college through graduate school as well as professional programmers. Subsequent studies could focus more closely on certain experience groups from this study or, more importantly, other experience groups such as K–12 learners.

#### **4.2.13 Data Diagnostics**

Participants were limited to adults (age 18+) either in a computer science program at the university level or professional programmers. All participants who completed the entire instrument were included in the data analysis. The majority of participants whose data was not included never answered the consent question nor any of the demographics questions; one participant selected "No, I Do Not Accept" to the consent question. The participants that consented to the study and answered the demographics questions were asked eight total questions with multiple sub-questions. Qualtrics required participants to complete each task before moving on to the next task. Inside of each task, each tool had to be categorized as "helpful" or "not helpful" in order for participants to move to the next question. Abandoned and incomplete assessments were excluded from the diagnostics, so sixteen participants that started answering the study but did not complete it were excluded.

#### **4.2.14 Analytic Strategy**

In this study, each scenario was considered to be independent. Participants were directed to categorize tools as helpful or not helpful, and we coded the selection of "not helpful" to zero. They were then asked to rank the helpful items from most helpful (first = 1) to least helpful (last). In the analysis, we did a reverse weighting of tools where the most helpful tool received the highest weight (16), the second most helpful received the next highest weight (15), and so on. This differentiated the most helpful tools from the tools that were categorized as "not helpful", which were assigned a weight of zero. Weighting the tools this way did not result in weights from 16 down to one; only the tools that were categorized as helpful



received weights. Thus, if a participant only rated three tools as helpful, those three tools would receive weights of 16, 15, and 14 in order and then the rest of the tools, which were categorized as "not helpful", received a weight of zero. We performed an independent samples ANOVA on each scenario with the tool type and the participants' experience level as interaction terms to determine if these affected the resulting weight. We also took means of the participants' tool weightings and ranked the tools' mean weight per scenario. In the analysis below, a table reporting the means and standard deviation is presented for each tool in order and the top three mean weights are bolded for each scenario.

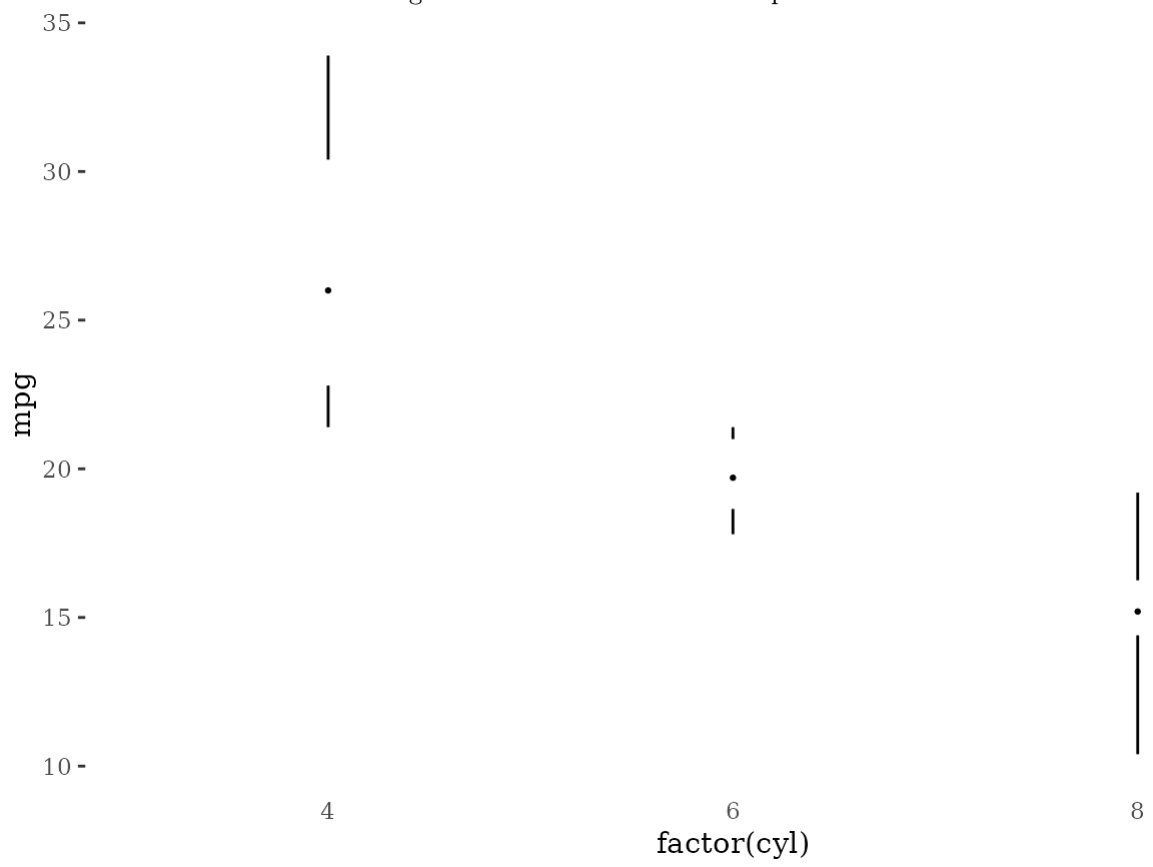
With the data that is reported in the sections below, there is a wide variance for the weightings of the tools. In order to visually represent data with a wide variance, we created Tufte box plots, to visually assess the weightings. An example from the online documentation [153], figure 4.2 is provided to help explain Tufte plots. In Tufte plots, the point signifies the median weight, the empty space is the interquartile range, and the lines are the whiskers. Compared to a traditional box and whisker plot, Tufte visualization helps to highlight the median and differentiate the weightings, since there is no box to overcrowd the visuals. In the example in figure 4.2, the median for "factor 4" is approximately 26, the median for "factor 6" is approximately 20, and the median for "factor 8" is approximately 15. An important aspect to keep in mind when viewing the Tufte box plots in the sections below is the point where the median is drawn. If the point is on the x-axis, then it means over half of the programmers categorized the tool as "not helpful" in the given scenario, which lowers the perceived value of the tool in that scenario.

## 4.3 Results

### 4.3.1 Participant Flow

Participants were recruited and took part in the study over a 35-day period in January and February, 2024. Table 4.1 lists the participant tallies per experience group, for a total of 123 participants across all of the experience levels. The study only allowed participants to use a desktop or laptop computer, so participants who attempted to access the instrument on mobile were notified it was not available and the attempt ended. A total of 333 participants initially attempted to take part in the study. One participant did not consent to take the study. A total of sixteen participants started the study but did not complete it. The remaining 123 participants completed all eight questions in the study, and all data they provided was analyzed for this work.

Figure 4.2: Tufté Box Plot Example



### 4.3.2 Recruitment

Recruiting for professional participants who were not enrolled in college was done via four emails in the first two weeks of January, 2024. University-level participants were recruited in person on four days at five total classes during the last week of January and the first full week of February. Responses started after the first email response and continued into the second full week of February. There was no follow-up to the same recruitment group after the initial recruitment period.

### 4.3.3 Statistics and Data Analysis

We utilized a mixed two-way analysis of variance (ANOVA) to analyze each scenario with the within-subjects predictor of *tool* type and between-subjects predictor of *experience* along with the requisite interaction term. The models estimate the relationship between predictors to arrive at the outcome variable, weight of the tool. In each case, we ensured the assumptions of a two-way ANOVA were met prior to running the ANOVA. When running the ANOVA, we chose to use Type II, which Langsrud [144] finds to be preferable to Type III for such analysis.

One of the fundamental assumptions in the ANOVA with a within-subjects procedure is that of sphericity, which checks whether the variance/covariance matrix of the observed data from the Repeated Measures follows a particular pattern. If Mauchly's test of sphericity indicates the assumption of sphericity is violated, we apply a Greenhouse-Geisser correction to the degrees of freedom used to calculate the F-ratio. These corrections often increase the p-value, so significance is reported again. Statistics were conducted and charts were created using RStudio Version 2023.12.1+402.

## Question 1: Use Statement

Figure 4.3: Question 1: Which Tools Would Be Helpful?

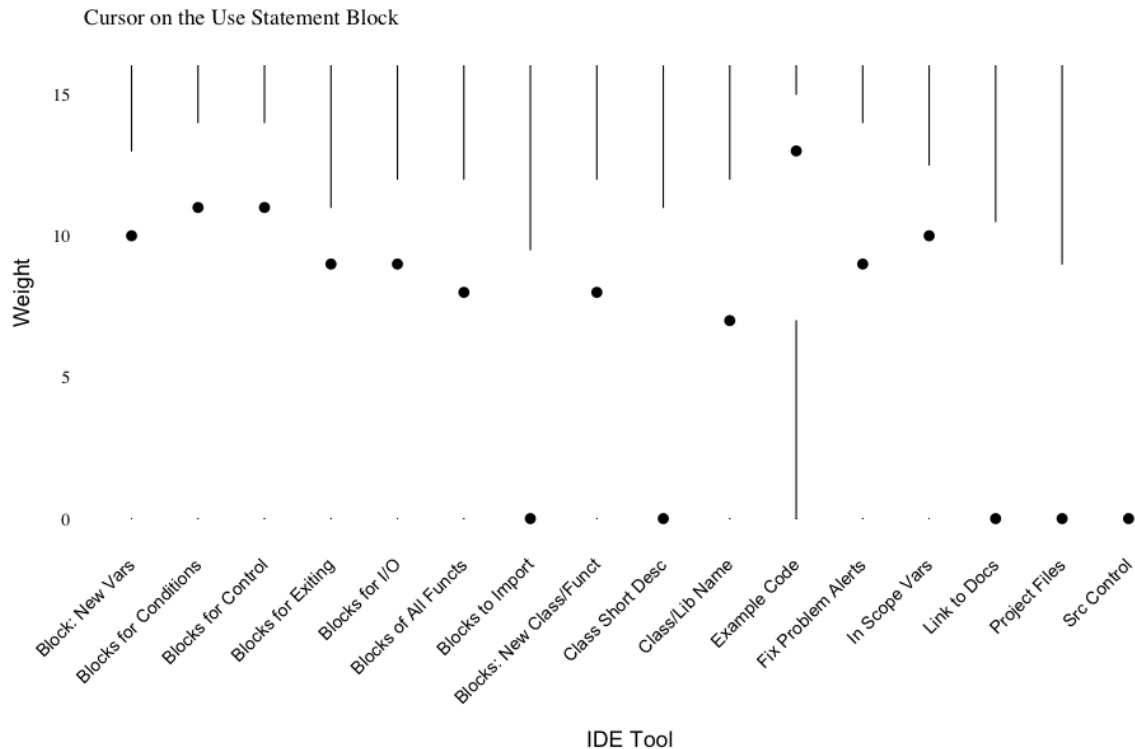


Table 4.3: Question 1: Use Statement Means/SD

Rank	Tool	<i>M</i>	<i>SD</i>
1	Example Code	10.22	5.89
2	Blocks for Conditionals (If / Else, etc)	8.87	6.06
3	Blocks for Control (Repetition / Loops)	8.80	6.08
4	Alert About What Problems Mean	8.29	6.43
5	Blocks to Create New Variables	8.24	5.68
6	Blocks of In-Scope Variables	7.64	5.95
7	Blocks for I/O (Print, Input, Clicks, etc)	7.33	5.66
8	Blocks for New Class/Funct	6.82	6.12
9	Blocks for Exiting (Return, Continue, Break, etc)	6.76	5.52
10	Blocks of All Functions in the Class	6.59	5.95
11	Class Name/Library	6.12	6.17
12	Class Short Description	4.96	6.07
13	Link to Class Documentation	4.71	6.10
14	Blocks to Import Additional Libraries (use)	4.18	5.75
15	Project Files / Structure	4.03	5.71
16	Source Control	2.40	4.63

The ANOVA for the scenario where the cursor is on a “use” statement is statistically significant for *tool*  $F(15,1815) = 15.75$ ,  $p < .001$ , generalized Eta squared ( $\eta^2G$ ) = .111, which demonstrates that participants rated the helpfulness of the tools significantly differently. Additional significance was found in *experience*  $F(1,121) = 4.59$ ,  $p = .034$ ,  $\eta^2G = .001$  and the *tool:experience* interaction  $F(15,1815) = 2.02$ ,  $p = .012$ ,  $\eta^2G = .016$ . The generalized Eta squared indicated the effect size of *tool* to be 11.1% of the model, ( $\eta^2G = .111$ ). The effect sizes of *experience* and the interaction of *tool:experience* are small at 0.1% and 1.6% respectively.

When analyzing Mauchly’s test, the data indicated the sphericity assumption was not met for the *tool:experience* interaction ( $W = .20$ ,  $p < .001$ ), so a Greenhouse-Geiser correction was applied ( $\epsilon = 0.55$ ). The *tool:experience* interaction was still found to be significant ( $p = .036$ ). The mean weights for each of the tools is in table 4.3. The top three mean weights in this scenario are “Example Code”, “Blocks for Conditionals”, and “Blocks for Control.” In figure 4.5, a rendering of the top tools is shown as an example. The top-ranked tools not shown were excluded due to the inability to display them at this stage, such as “Alert About What Problems Mean,” because there are no errors. A context-aware palette would not show inconsequential information.

Figure 4.4: Question 1: Code Snippet Displayed to Participant

```
1 use Libraries.Containers.Array ←  
2 use Libraries.Containers.Iterator
```

Figure 4.5: Question 1: Potential Partial Palette Based on Weights

Projects Scene **Palette**

[See Example Code](#)

```
if true  
end  
elseif true  
else  
repeat 1 times  
end  
repeat while false  
end  
// my comment  
integer a = 0  
number b = 0.0  
boolean bool = true  
text string = "words"  
output "words"  
say "words"  
action myAction (type name)  
end
```

## Question 2: Syntax Error

Figure 4.6: Question 2: Which Tools Would Be Helpful?

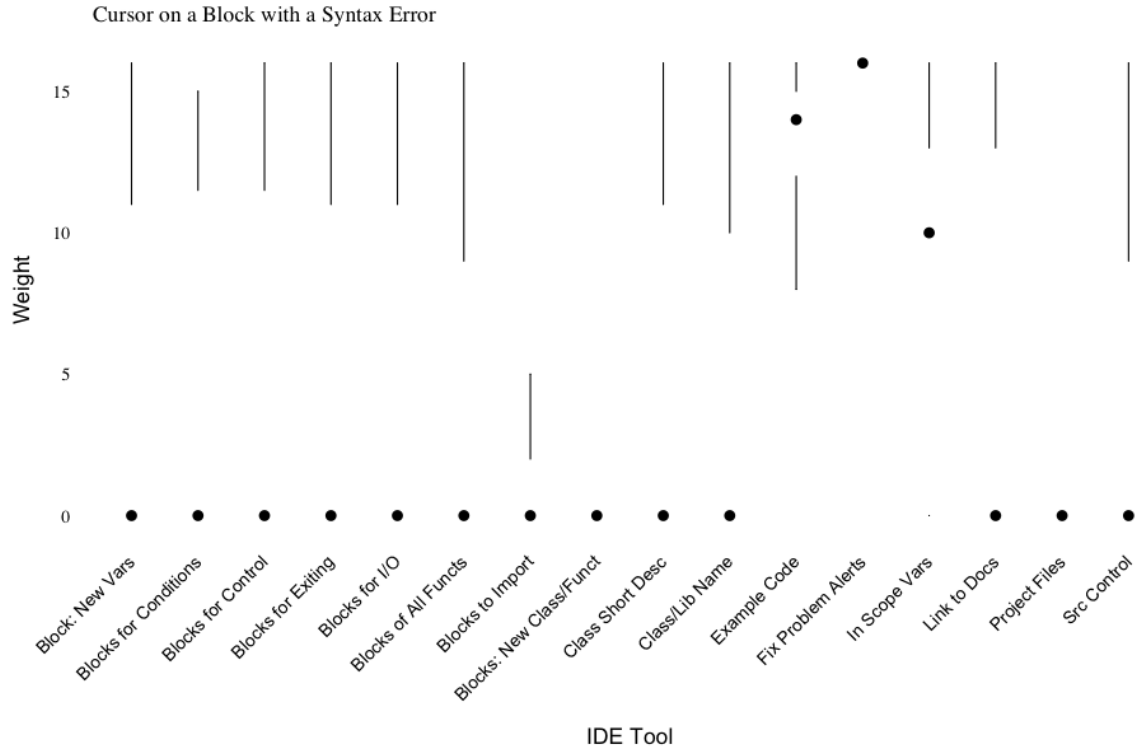


Table 4.4: Question 2: Syntax Error Means/SD

Rank	Tool	<i>M</i>	<i>SD</i>
1	Alert About What Problems Mean	15.09	2.44
2	Example Code	12.04	4.99
3	Blocks of In-Scope Variables	7.54	6.29
4	Link to Class Documentation	5.71	6.35
5	Blocks for I/O (Print, Input, Clicks, etc)	5.37	5.88
6	Blocks for Control (Repetition / Loops)	5.32	6.13
7	Blocks for Exiting (Return, Continue, Break, etc)	5.06	5.70
8	Blocks for Conditionals (If / Else, etc)	4.80	5.81
9	Blocks to Create New Variables	4.67	5.80
10	Class Short Description	4.23	6.04
11	Class Name/Library	3.97	5.85
12	Source Control	3.64	5.68
13	Blocks of All Functions in the Class	3.63	5.49
14	Project Files / Structure	2.63	5.01
15	Blocks to Import Additional Libraries (use)	2.57	4.64
16	Blocks for New Class/Funct	2.48	4.89

The ANOVA for the scenario where the cursor is on a red block, which indicates a syntax error, is statistically significant for *tool*  $F(15,1815) = 49.34$ ,  $p < .001$ ,  $\eta^2G = .271$ , which demonstrates that participants rated the helpfulness of the tools significantly differently. The generalized Eta squared indicated the effect size of *tool* to be relatively large at 27.1% of the model, ( $\eta^2G = .271$ ). No other significance was found in this scenario.

When analyzing Mauchly's test, the data indicated the sphericity assumption was not met for the tool type ( $W = .01$ ,  $p < .001$ ), so a Greenhouse-Geiser correction was applied ( $\epsilon = 0.55$ ). The tool type was still found to be significant ( $p < .001$ ). The mean weights for each of the tools is in table 4.4. The top three mean weights in this scenario are "Alert About What Problems Mean", "Example Code", and "Blocks of In-Scope Variables." In figure 4.8, a rendering of the top tools is shown as an example. Only the top-ranked tools with a median higher than zero are rendered, and there are no "In Scope Variables" in this example, which is why there are only two tools in the example palette.



Figure 4.7: Question 2: Code Snippet Displayed to Participant

```
1 use Libraries.Containers.Iterator
2 class Main
3   action PrintEven ( Array<integer> array )
4
5   end
6 end
```

Figure 4.8: Question 2: Potential Partial Palette Based on Weights



### Question 3: If Statement

Figure 4.9: Question 3: Which Tools Would Be Helpful?

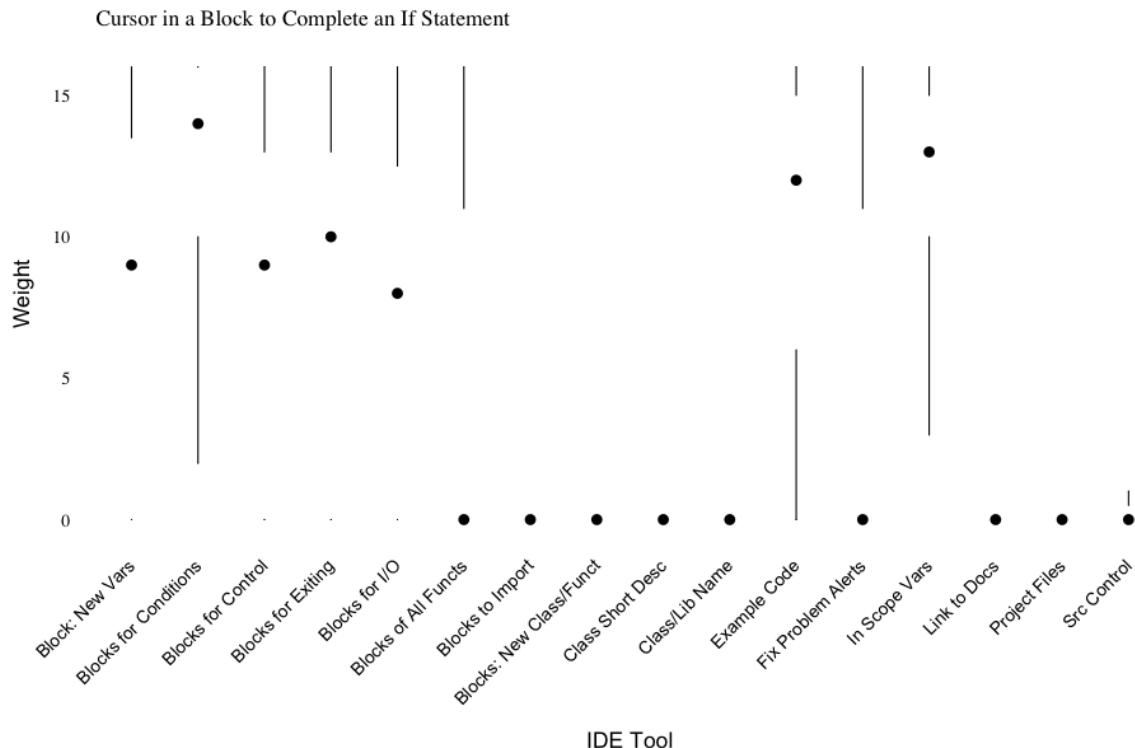


Table 4.5: Question 3: If Statement Means/SD

Rank	Tool	<i>M</i>	<i>SD</i>
1	Blocks for Conditionals (If / Else, etc)	11.27	6.04
2	Blocks of In-Scope Variables	11.16	5.72
3	Example Code	10.00	6.00
4	Blocks for Exiting (Return, Continue, Break, etc)	7.29	6.34
5	Blocks to Create New Variables	6.89	6.51
6	Blocks for Control (Repetition / Loops)	6.72	6.44
7	Blocks for I/O (Print, Input, Clicks, etc)	6.28	6.27
8	Blocks of All Functions in the Class	4.54	6.00
9	Alert About What Problems Mean	4.07	6.01
10	Link to Class Documentation	2.82	5.28
11	Class Short Description	2.74	5.15
12	Source Control	2.59	4.97
13	Project Files / Structure	2.41	4.99
14	Class Name/Library	2.28	4.71
15	Blocks for New Class/Funct	2.23	4.82
16	Blocks to Import Additional Libraries (use)	1.48	3.86

The ANOVA for the scenario where the cursor is inside a block for the user to type the conditions for the “if” statement is statistically significant for *tool*  $F(15,1815) = 44.09$ ,  $p < .001$ ,  $\eta^2G = .252$ , which demonstrates that participants rated the helpfulness of the tools significantly differently. Additional significance was found in *experience*  $F(1,121) = 18.78$ ,  $p < .001$ ,  $\eta^2G = .011$ . The generalized Eta squared indicated the effect size of tool preference to be relatively large at 25.2% of the model, ( $\eta^2G = .252$ ). The effect size of *experience* is small at 1.1% of the model ( $\eta^2G = .011$ ).

When analyzing Mauchly’s test, the data indicated the sphericity assumption was not met for the tool preference ( $W = .04$ ,  $p < .001$ ), so a Greenhouse-Geiser correction was applied ( $\epsilon = 0.68$ ). The tool preference was still found to be significant ( $p < .001$ ). The mean weights for each of the tools is in table 4.5. The top three mean weights in this scenario are “Blocks for Conditionals”, “Blocks of In-Scope Variables”, and “Example Code.” In figure 4.11, a rendering of the top tools is shown as an example. All of the top-ranked tools with a median higher than zero are rendered in the example.

Figure 4.10: Question 3: Code Snippet Displayed to Participant

```

1  use Libraries.Containers.Array
2  use Libraries.Containers.Iterator
3  class Main
4      action PrintEven (Array<integer> array)
5          Iterator<integer> it = array:GetIterator()
6          repeat while it:HasNext()
7              integer value = it:Next()
8              if
9                  end
10             end
11         end
12     end
13 end
14 action Main
15     end
16 end
17 end

```

Figure 4.11: Question 3: Potential Partial Palette Based on Weights

Projects
Scene
Palette

if true
end

elseif true
else

array

it

value

[See Example Code](#)

return value

// my comment

integer a = 0

number b = 0.0

boolean bool = true

text string = "words"

repeat 1 times
end

repeat while false
end

output "words"

say "words"

## Question 4: Add Function

Figure 4.12: Question 4: Which Tools Would be Helpful?

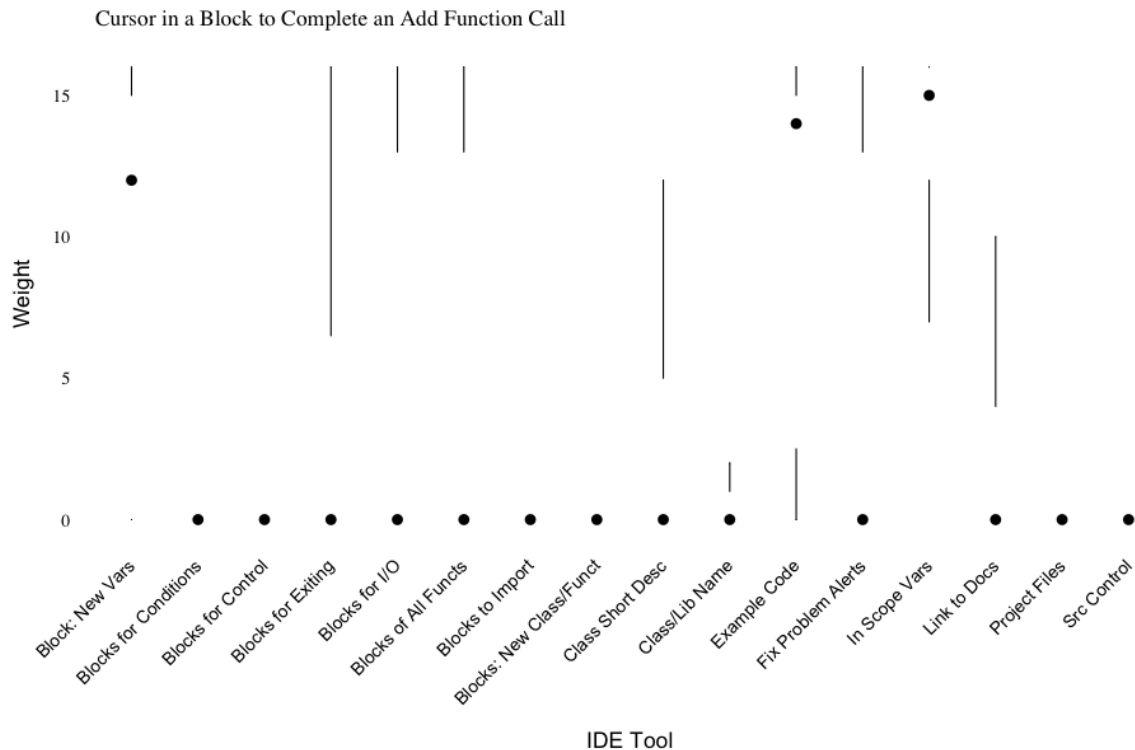


Table 4.6: Question 4: Add Function Means/SD

Rank	Tool	$M$	$SD$
1	Blocks of In-Scope Variables	12.20	5.64
2	Example Code	10.33	6.40
3	Blocks to Create New Variables	7.98	7.08
4	Blocks of All Functions in the Class	6.15	6.82
5	Alert About What Problems Mean	5.35	6.57
6	Blocks for I/O (Print, Input, Clicks, etc)	4.89	6.52
7	Link to Class Documentation	3.11	5.44
8	Blocks for Exiting (Return, Continue, Break, etc)	3.08	5.36
9	Class Short Description	3.07	5.37
10	Class Name/Library	2.98	5.37
11	Blocks for Conditionals (If / Else, etc)	2.94	5.32
12	Blocks for New Class/Funct	2.51	5.20
13	Project Files / Structure	2.49	4.95
14	Blocks for Control (Repetition / Loops)	2.32	4.77
15	Source Control	2.16	4.71
16	Blocks to Import Additional Libraries (use)	1.77	4.45

The ANOVA for the scenario where the cursor is inside a block for the user to fill out the parameters in the "Add()" function call is statistically significant for *tool*  $F(15,1815) = 38.03$ ,  $p < .001$ ,  $\eta^2G = .223$ . No other significance was found. The generalized Eta squared indicated the effect size of *tool* to be relatively large at 22.3% of the model, ( $\eta^2G = .223$ ).

When analyzing Mauchly's test, the data indicated the sphericity assumption was not met for tool type ( $W = .05$ ,  $p < .001$ ), so a Greenhouse-Geiser correction was applied ( $\epsilon = 0.71$ ). The tool type was still found to be significant ( $p < .001$ ). The mean weights for each of the tools is in table 4.6. The top three mean weights in this scenario are "Blocks of In-Scope Variables", "Example Code", and "Blocks to Create New Variables." Figure 4.14 shows a rendering of the top tools from this example. The top-ranked tools with a median higher than zero are rendered in the example. In this context, "Blocks of In-Scope Variables" was not rendered for clarity because there are no existing variables in scope in this example. In cases where there are variables in-scope, that tool should be displayed right after "Example Code."

Figure 4.13: Question 4: Code Snippet Displayed to Participant

```
1  action Main
2  Add( )
3  end
4  action Add (integer a, integer b)
5  output a + b
6  end
```

Figure 4.14: Question 4: Potential Partial Palette Based on Weights

Projects Scene **Palette**

[See Example Code](#)

```
// my comment
```

```
integer a = 0
```

```
number b = 0.0
```

```
boolean bool = true
```

```
text string = "words"
```

## Question 5: Dataframe

Figure 4.15: Question 5: Which Tools Would be Helpful?

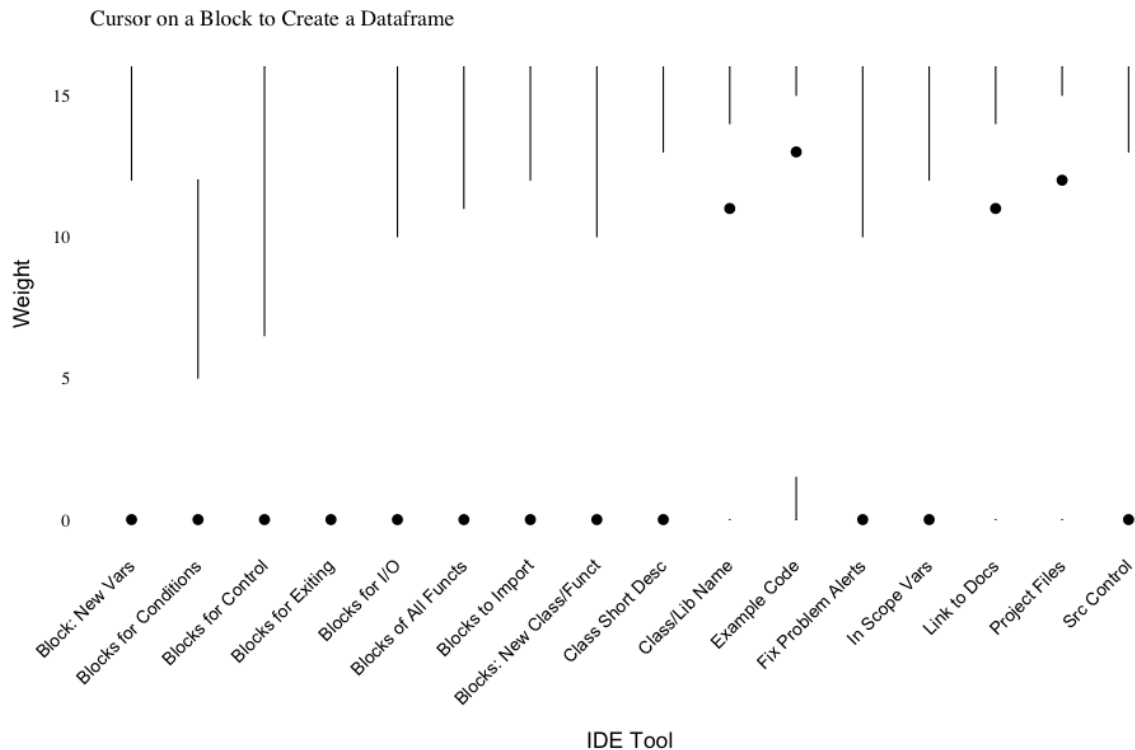


Table 4.7: Question 5: Create Dataframe Means/SD

Rank	Tool	<i>M</i>	<i>SD</i>
1	Example Code	9.80	6.15
2	Project Files / Structure	8.58	6.84
3	Class Name/Library	7.87	6.76
4	Link to Class Documentation	7.80	6.59
5	Class Short Description	6.13	6.64
6	Blocks to Import Additional Libraries (use)	5.85	6.41
7	Blocks of In-Scope Variables	5.75	6.18
8	Source Control	5.72	6.51
9	Blocks to Create New Variables	5.50	6.15
10	Blocks of All Functions in the Class	4.91	6.12
11	Blocks for New Class/Funct	4.47	5.99
12	Alert About What Problems Mean	4.40	6.01
13	Blocks for I/O (Print, Input, Clicks, etc)	4.19	6.07
14	Blocks for Control (Repetition / Loops)	3.05	5.28
15	Blocks for Conditionals (If / Else, etc)	2.89	5.10
16	Blocks for Exiting (Return, Continue, Break, etc)	1.82	4.01



Figure 4.16: Question 5: Code Snippet Displayed to Participant

```

1 use Libraries.Compute.Statistics.DataFrame
2 use Libraries.Interface.Controls.Charts.Histogram
3 // Create a DataFrame to hold the data.
4 DataFrame frame ←
5 // Load your data file into the frame.

```

Figure 4.17: Question 5: Potential Partial Palette Based on Weights

Projects	Scene	Palette
<a href="#">See Example Code</a> Dataframe Class <a href="#">Read the Documentation</a>		

The ANOVA for the scenario where the cursor is on a “Dataframe” instantiation is statistically significant for *tool*  $F(15,1815) = 16.05, p < .001, \eta^2G = .109$ . Additional significance was found in the *tool:experience* interaction  $F(15,1815) = 3.48, p < .001, \eta^2G = .026$ . The generalized Eta squared indicated the effect size of *tool* to be 10.9% of the model, ( $\eta^2G = .109$ ). The effect size of the *tool:experience* interaction is small at 2.6%.

When analyzing Mauchly’s test, the data indicated the sphericity assumption was not met for tool type ( $W = .02, p < .001$ ) nor the *tool:experience* interaction ( $W = .02, p < .001$ ), so a Greenhouse-Geiser correction was applied ( $\epsilon = 0.65$ ). The tool type was still found to be significant ( $p < .001$ ). Likewise, the *tool:experience* interaction was also found to be significant ( $p < .001$ ). The mean weights for each of the tools is in table 4.7. The top three mean weights in this scenario are “Example code”, “Project Files / Structure”, and “Class Name/Library.” In figure 4.17, a rendering of the top tools is rendered for this example. The top-ranked tools with a median higher than zero are rendered in the example. The “Project Files / Structure” is rendered as the “Projects” tab.

## Question 6: Chart Display

Figure 4.18: Question 6: Which Tools Would be Helpful?

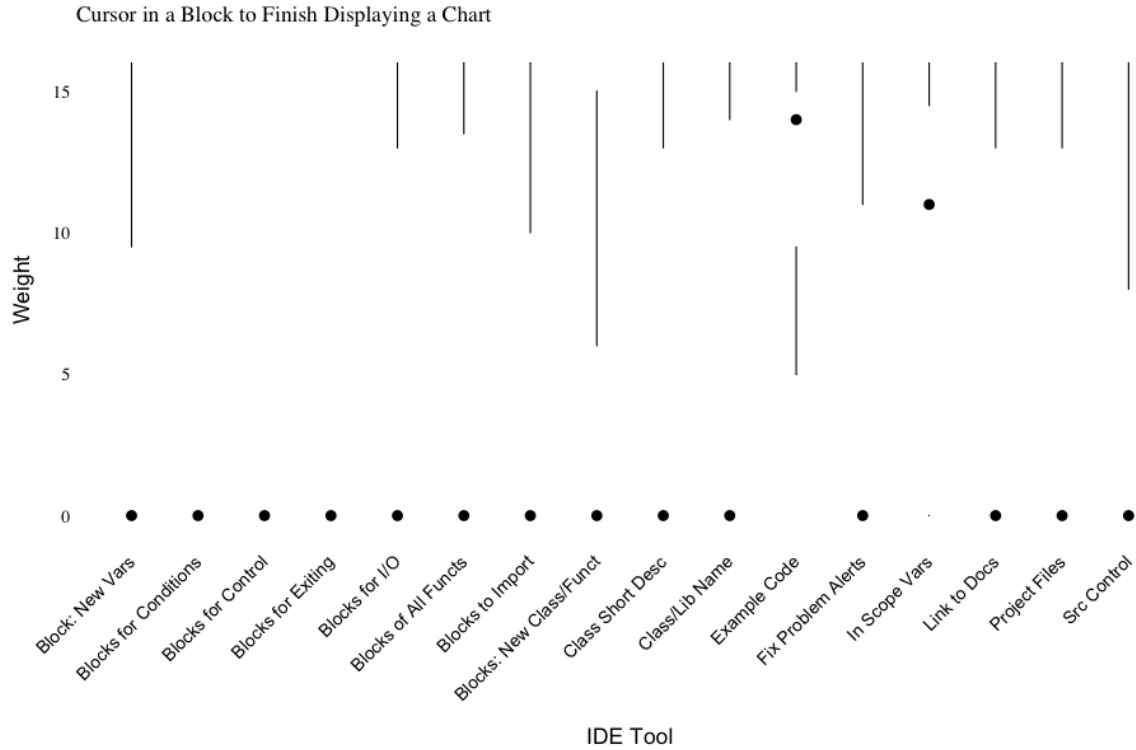


Table 4.8: Question 6: Chart Display Means/SD

Rank	Tool	$M$	$SD$
1	Example Code	11.15	5.62
2	Blocks of In-Scope Variables	7.80	6.80
3	Class Name/Library	6.34	6.81
4	Blocks of All Functions in the Class	6.20	6.92
5	Link to Class Documentation	6.09	6.49
6	Blocks for I/O (Print, Input, Clicks, etc)	5.67	6.74
7	Project Files / Structure	5.55	6.66
8	Class Short Description	5.35	6.36
9	Alert About What Problems Mean	4.65	6.18
10	Blocks to Create New Variables	3.89	5.70
11	Blocks to Import Additional Libraries (use)	3.80	5.67
12	Source Control	3.58	5.76
13	Blocks for New Class/Funct	3.23	5.64
14	Blocks for Conditionals (If / Else, etc)	2.63	5.18
15	Blocks for Control (Repetition / Loops)	2.19	4.75
16	Blocks for Exiting (Return, Continue, Break, etc)	2.14	4.84

The ANOVA for the scenario where the cursor is on line to display a “chart” is statistically significant for *tool*  $F(15,1815) = 19.14$ ,  $p < .001$ ,  $\eta^2G = .126$ . Additional significance was found in the *tool:experience* interaction  $F(15,1815) = 5.07$ ,  $p < .001$ ,  $\eta^2G = .037$ . The generalized Eta squared indicated the effect size of *tool* to be 12.6% of the model, ( $\eta^2G = .126$ ). The effect size of the *tool:experience* interaction is small at 3.7%.

When analyzing Mauchly’s test, the data indicated the sphericity assumption was not met for tool type ( $W = .04$ ,  $p < .001$ ) nor the *tool:experience* interaction ( $W = .04$ ,  $p < .001$ ), so a Greenhouse-Geiser correction was applied ( $\epsilon = 0.68$ ). The tool type was still found to be significant ( $p < .001$ ). Likewise, the *tool:experience interaction* was also found to be significant ( $p < .001$ ). The mean weights for each of the tools is in table 4.8. The top three mean weights are “Example code”, “Blocks of In-Scope Variables”, and “Class Name/Library.” In figure 4.20, a rendering of the top tools is rendered for this example. The top-ranked tools with a median higher than zero, of which there were only two, are rendered in the example.

## 4.4 Discussion

### 4.4.1 Support of Original Hypotheses

Through this study, we gained insight into what IDE tools might be helpful in an array of scenarios, at least for the purposes of creating an initial data-driven prototype. Some of the findings are more applicable to IDEs broadly rather than being specific to block languages, which we found interesting.

### 4.4.2 Hypothesis 1

*H<sub>0</sub>: Professionals and novices will consider all tools to provide the same helpfulness.*

The helpfulness of various IDE tools was rated significantly differently with a strong effect size, thus we reject the null hypothesis 1 in favor of the alternative hypothesis: there is a difference in which tools are helpful or not regardless of the scenario. This finding was supported for both novices and professionals. When analyzing the mean weights, we see that some tools are generally helpful across various scenarios. The tools that are ranked most often in the top three while never appearing in the bottom three are “Example Code” and “Blocks of In-Scope Variables.” In addition, the “Alert About What Problems Mean” and “Blocks to Create New Variables” are never in the bottom three rankings. Overall, this indicates these tools are generally helpful for programming tasks. We additionally see some tools as not helpful in most use cases. The tools “Blocks for Exiting,” “Blocks to Import Additional Libraries,” and “Source Control”

Figure 4.19: Question 6: Code Snippet Displayed to Participant

```
1 use Libraries.Compute.Statistics.DataFrame
2 use Libraries.Interface.Controls.Charts.Histogram
3 // Create a DataFrame to hold the data.
4 DataFrame frame
5 // Load your data file into the frame.
6 frame:Load("data/AB_NYC_2019.csv")
7 // Select data from the numerical column "price" for the histogram.
8 frame:AddSelectedColumns("price")
9 // Using the frame, create a Histogram object.
10 Histogram chart = frame:Histogram()
12 chart: ←
```

Figure 4.20: Question 6: Potential Partial Palette Based on Weights

Projects Scene **Palette**

[See Example Code](#)

chart

frame

are never rated in the top three tools.

### 4.4.3 Hypothesis 2

*H<sub>0</sub>: Professionals and novices will want the same tools regardless of the scenario.*

Tools were rated significantly differently in each scenario, thus we reject the null hypothesis 2 in favor of the alternative hypothesis: depending on the scenario, some tools were more useful than others. When looking across the scenarios as a whole, we recognize some phases of programming.

Question one with the “use statement” represents a typical scenario for the beginning phase of writing a program. This is a time when the programmer is starting out and needs some early assistance to put the structure together. In this scenario, the highest ratings were for “Example Code,” “Blocks for Conditionals,” and “Blocks for Control,” which is unsurprising given that the programmer needs to know how to begin solving the programming challenge and to begin their programmatic scaffolding. Interestingly, “Blocks to Create New Variables” was in the top five tools; we expected this tool to be ranked slightly higher. Additionally “Blocks to Import Additional Libraries” was ranked in the bottom three, although we expected it to be ranked in the top three. When looking at figure 4.3, we see that the median weights of most tools are non-zero, which reinforces the idea that the programmers want a variety of tools to begin their programming tasks. The tools with a zero median, meaning at least half of the developers found them not helpful, are “Blocks to Import Additional Libraries,” “Class Short Description,” “Link to Docs,” “Project Files,” and “Source Control.”

When problems arise, such as in the case when there is a red block due a syntax error, programmers reached for the top two tools to figure out how to fix the error—“Alert About What Problems Mean” and “Example Code”—which is as we expected. What is interesting is that the third highest-ranked tool was “Blocks of In-Scope Variables,” which would not affect this situation, but the actual fix for it—“Blocks to Import Additional Libraries,” namely the array library in this case— was the second-lowest-ranked tool. In this scenario, the only non-zero medians (as shown in figure 4.6) are the same blocks as the top three means. This indicates that for an error, over half of the developers found nothing else helpful outside of figuring out how to fix the error.

In the midst of programming, it is common to write “if statements” as well as create and call functions, which we tested in questions three and four respectively. In both of these scenarios, “Blocks of In-Scope Variables” and “Example Code” were ranked in the top three tools. “Blocks for Conditionals” was the top-ranked tool for the question about “if statements,” which is not surprising considering developers might

want to add “else if” and “else statements.” For the add statement, “Blocks to Create New Variables” rounds out the top three tools. Since in this case the cursor indicated that the user needed to add a parameter, it makes sense that the programmer would want both the option to use currently in-scope variables and to create a new one. In figure 4.9, there are seven tools that over half of the programmers rated as helpful: the top three ranked tools plus “Blocks to Create New Variables,” “Blocks for Control,” “Blocks for Exiting,” and “Blocks for I/O.” When analyzing the medians in figure 4.12 for the “Add()” function, we see that over half of the programmers rated only the top three tools as helpful.

The last two scenarios related to dataframes and charts give us an indication of what programmers need when they encounter what we consider to be an unfamiliar scenario. In both of these scenarios, the top-ranked tool was “Example Code.” which indicates programmers are simply trying to learn what it is they are supposed to be doing. In both scenarios, the third-highest-ranked tool was “Class Name/Library,” but this might be misleading. As shown in figure 4.18 for scenario six, regarding displaying a chart, over half of the programmers categorized “Class Name/Library” as not helpful. The other top-three tools in each scenario, respectively, were “Blocks of In-Scope Variables” and “Project Files / Structure.” Looking at the medians in figure 4.15, we see “Link to Class Documentation” tool was the only other tool selected as helpful by over half of the programmers for the question about dataframes. In figure 4.18, only the two top-ranked tools were categorized as helpful by over half of the programmers. Interestingly, in both scenarios, the three lowest-ranked tools were “Blocks for Control,” “Blocks for Conditionals,” and “Blocks for Exiting.” These results indicate that programmers did not know how to approach these problems, but at a high level they had an idea of what they did not need for these scenarios: loops, “if statements,” or return statements.

#### 4.4.4 Hypothesis 3

*H<sub>0</sub>: Professionals and novices will have the same viewpoint on the helpfulness of the tools.*

Experience level was only significantly different in some of the scenarios, and even when it was, the effect size was typically less than one percent of the model. This was likely due in part to the small sample of professional programmers and should be considered for a future study. Considering there was no significant difference in most scenarios, we fail to reject the null hypothesis 3.

Table 4.9: Count of Times Each Tool Ranks in the Top or Bottom Tools Across All Scenarios

#	Tool Type	Ranked in Top 3	Ranked in Bottom 3
1	Project Files/Structure	1	2
2	Source Control	0	2
3	Class Name/Library	2	1
4	Class Short Description	0	1
5	Blocks of In-Scope Variables	4	0
6	Alert About What Problems Mean	1	1
7	Link to Class Documentation	0	0
8	Blocks to Create New Variables	1	0
9	Blocks for Control	1	3
10	Blocks for Conditionals	2	2
11	Blocks of All Functions in the Class	0	0
12	Blocks for I/O	0	0
13	Blocks to Create New Blocks	0	2
14	Blocks for Exiting	0	2
15	Blocks to Import Additional Libraries	0	4
16	Example Code	6	0

#### 4.4.5 Similarity of Results

As discussed previously, Dwyer et al. [87] and Gaver [150] recommended researchers to consider all visual cues and affordances when creating block-based IDEs. Mason and Cooper [154] report that having extra options, tools, or affordances available in an IDE reduces performance, especially in the learning environment, even if they are not used. This in turn causes novices’ perception that programming in learning and professional environments is more difficult. Essentially, they state that novices benefit from simplified environments. The current study provides quantitative analysis to expand upon these recommendations; in each example a minimal context-aware palette is provided.

#### 4.4.6 Interpretation

This study aimed to quantify which IDE tools would be helpful or not helpful in various scenarios. Table 4.9 shows how many times each tool was ranked in the top three or bottom three. Table 4.11 shows the number of times each tool had a median greater than zero, meaning that more than half of the programmers categorized the tool as helpful for the scenario. Many times, the median is zero, which means that over half of the programmers categorized the tool as not helpful for the given scenario. Three tools were not found to be helpful by more than half of the developers in any of the scenarios: “Class Short Description,” “Source Control,” and “Blocks to Import Additional Libraries.”

Table 4.10: Sum of the Means

Rank	Tool	Mean Sum
1	Example Code	63.54
2	Blocks of In-Scope Variables	52.09
3	Alert About What Problems (Red Blocks) Mean	41.85
4	Blocks to Create New Variables	37.18
5	Blocks for I/O (Print, Input, Clicks, etc)	33.72
6	Blocks for Conditionals (If, Else, etc)	33.41
7	Blocks of All Functions in the Class	32.03
8	Link to Class Documentation	30.24
9	Class Name/Library	29.56
10	Blocks for Control (Repetition, Loops)	28.40
11	Class Short Description	26.47
12	Blocks for Exiting (Return, Continue, Break, etc)	26.15
13	Project Files / Structure	25.69
14	Blocks to Create New Blocks (Class, Function, etc)	21.74
15	Source Control	20.09
16	Blocks to Import Additional Libraries (use)	19.64

Table 4.11: Tool Median Weight Greater Than Zero

Rank	Tool	Q1 Use	Q2 Error	Q3 If	Q4 Add()	Q5 Datafrm	Q6 Chart	Total
1	Example Code	✓	✓	✓	✓	✓	✓	6
2	Blocks of In-Scope Variables	✓	✓	✓	✓		✓	5
3	Alert About What Problems Mean	✓	✓					2
4	Blocks to Create New Variables	✓		✓	✓			3
5	Blocks for I/O	✓		✓				2
6	Blocks for Conditionals	✓		✓				2
7	Blocks of All Functions in the Class	✓						1
8	Link to Class Documentation					✓		1
9	Class Name/Library	✓				✓		2
10	Blocks for Control	✓		✓				2
11	Class Short Description							0
12	Blocks for Exiting	✓		✓				2
13	Project Files / Structure					✓		1
14	Blocks to Create New Blocks	✓						1
15	Source Control							0
16	Blocks to Import Additional Libraries							0
Total per scenario		11	3	7	3	4	2	



When assessing across scenarios, we discover that certain tools regularly emerge as most helpful regardless of scenario. The “Example Code” tool was ranked in the top three tools and was categorized as helpful by over half of the programmers in every scenario, so having a way to display or link to example code would be beneficial to programmers. The “Blocks of In-Scope Variables” tool was ranked in the top three tools in four of the six scenarios, so in the majority of cases, providing this tool would be beneficial for writing programs. This tool was only ranked outside the top three on the “use statement” (question one), likely due to the fact that programmers could recognize there were not yet any local scope variables, but it was still ranked as broadly helpful in that scenario according to the median. For the dataframe question, that same tool was categorized as not helpful by over half of the developers, which is likely because “dataframe” is an unknown term or programming scenario for many programmers, hence they ranked tools that provided them more information highest in that scenario. Neither of these tools (“Example Code” nor “Blocks of In-Scope Variables”) were ever ranked in the bottom five rankings in any scenario, which indicates they would be helpful across most programming activities. These findings are reinforced if we look at table 4.10, where these tools are the top two sums by weight. These tools are of high benefit to users and should be included persistently in the context-aware palette with prominent placement when their existence is possible. (Cases in which their existence is not possible include a newly opened blank file.)

Conversely, we discovered some tools are consistently in the bottom of the rankings. As described above, “Class Short Description,” “Source Control,” and “Blocks to Import Additional Libraries” were never rated as helpful by over half of the programmers in any scenario. The “Source Control” tool was ranked in the bottom three twice, and it was ranked in the bottom five for all but one of the scenarios. The dataframe question was the only one in which this tool did not appear in the bottom five rankings. Programmers also ranked the tool “Blocks to Import Additional Libraries” in the bottom three tools in four of the six scenarios. None of these tools were ever ranked in the top three in any scenario, which collectively indicates they would not be broadly helpful across all programming activities. These findings are reinforced if we look at table 4.10, where two of the three tools are at the bottom by mean sums. These tools are of little benefit to users and should likely not be included in the context-aware palette.

Additional analysis is required to make recommendations for a context-aware palette. The rankings with corresponding weights in table 4.10 are one consideration for which tools to adopt in a context-aware palette. We also analyzed the top three and bottom three rankings in each scenario from table 4.9 and cross-referenced them with the rankings in the sum of the means in table 4.10 and the frequency with which the majority of developers categorized the tool as helpful in table 4.11.

As we move down the weighting order, the value of displaying tools in most use cases diminishes quickly. For example, “Example Code” and “Source Control” could be used in most scenarios, but the rankings for each are quite different, with comparative sums of means equaling 63.54 and 20.09 respectively. The difference in rankings from participants demonstrates that “Example Code” should be available as often as possible in the palette, whereas “Source Control” is less important and thus may not need to be available. Returning to the conclusions from Mason and Cooper [154], including a tool like “Source Control” may actually hinder learning and productivity; thus, we do not include it in any of our recommendations. There may be certain tools that are ranked low in table 4.10, yet other indicators point to them having high value in specific scenarios. For example, “Blocks for Exiting” was ranked low in the sums of means, but over half of the developers found it helpful in two scenarios. Similarly, “Project Files / Structure” was ranked even lower than “Blocks for Exiting,” but it was ranked in the top three in the fifth scenario regarding dataframes. In our renderings, “Project Files / Structure” is represented in a different tab than our context-aware palette.

“Example Code,” “Blocks of In-Scope Variables,” and “Blocks to Create New Variables” were the top tools categorized as helpful most often by developers, so they should be available whenever possible. The “Example Code” could be displayed as a link, pop-out, or another unobtrusive format, so that the length of the example code would not overtake the entire palette. In the renderings below, it is displayed as a link for ease of visualization. We must also take into account the visual and cognitive overhead each tool would present. For example, “Blocks of All Functions in the Class” is ranked before “Link to Class Documentation,” but for a robust class, there might be hundreds or thousands of functions that could be displayed. This takes compute cycles to parse and display as blocks and also presents a lot of visual information, which could be overwhelming for some users, especially novices, as previously discussed. Interestingly, neither of these tools were ever ranked in the top three nor bottom three, and they were only rated as helpful by the majority of programmers in one scenario each. A “Link to Class Documentation” may provide the ability to get just as much information without the overhead. We recommend having only one of these two tools available until more data can be gathered. If “Blocks of All Functions in the Class” is chosen, then it is recommended to list this at the end of the tool options and/or to provide a filter so as to not potentially overwhelm users. This is how this tool is represented in our example visualizations.

“Alert About What Problems Mean” is another top-ranked tool by sums of weights. This highly-specialized tool is most helpful when the IDE can detect a syntax error or some other issue in the code prior to compilation. We recommend to hide this tool until such a time that the IDE detects a syntax error.

In that scenario, the block containing the error should turn red, and when the cursor is on that block, this tool can appear, providing potential fixes. Until such a time, this tool does not need to be displayed, but when it is, it should appear near the top of the palette.

“Blocks for I/O” and “Blocks for Conditionals” are the next highly-ranked tools in the sum of weights. It makes sense that “Blocks for I/O” received a high ranking since questions one through three asked the participant to “Imagine you’ve been tasked to write a small function that will take in an integer array, *prints* out the even integers, and then returns an array of the even integers to the caller.” Thus, the participant was primed that they needed “Blocks for I/O” to *print* out the array. Likewise, “Blocks for Conditionals” likely received a high ranking because question three asked directly about “if statements” in the study. Due to the fact that control statements (repetitions like traditional while and for loops) are a similar fundamental concept as conditionals, it would follow that, despite their much lower ranking, “Blocks for Control” should be displayed below the “Blocks for Conditionals” tool.

Interestingly, the tool “Blocks for Exiting” received a low ranking, despite the previously-mentioned priming where the user was asked that it “...*returns an array* of the even integers” in questions one through three. This tool appears to have limited use due to its ranking in the sum of weights (twelfth) and its appearance in the top and bottom lists, where it was never ranked in the top three tools yet was twice ranked in the bottom three. The aforementioned fact that over half of the programmers categorized it as helpful in two scenarios does lend credence to support it existing on the palette at certain times. “Blocks to Create New Blocks” and “Blocks to Import Additional Libraries” sit at the bottom of the rankings. These tools could be displayed when a new project file is created and programmers need to begin importing libraries and creating new classes and functions, but the data from this specific study does not support their inclusion.

A true context-aware palette could have infinite arrangements depending on the amount of code in the file, where the cursor sits in the code, and the number of tools potentially available. In synthesis of the above, we will list the suggested palette tools and their order for context-aware palettes for the following examples:

1. Figure 4.21: when a new project file is created
2. Figure 4.22: editing of a script with code not in a class and there are no imported libraries
3. Figure 4.23: editing of a script with imported libraries
4. Figure 4.24: editing a file within code in a class

5. Figure 4.25: when there is an error in a program

For this first scenario in figure 4.21, the top three weighted tools of “Example Code,” “Blocks of In-Scope Variables,” and “Alert About What Problems Mean” are not valid tools to display. Without any code, the palette cannot decipher what example code to display, since there would be far too many options. There are no in-scope variables to show and there are no errors about which to alert the programmer, so neither of these tools can be displayed, either. In this case, the palette will display the most common blocks for the programmer to start writing code.

In this second scenario in figure 4.22, the “Example Code” tool is still not showing, since the palette does not have context for what examples to show, and again there are no errors to show, which means “Alert About What Problems Mean” is again not valid to display. The “Blocks of In-Scope Variables” tool is present with “my\_int” and “my\_counter” showing on the palette.

The primary difference we see in the scenario in figure 4.23 is the “Example Code” tool is now displayed as the most prominent tool. In order for this to be shown, the example assumes a library was imported to narrow the range of possible example code for the programmer.

Figure 4.24 demonstrates how the palette could look when editing in a class. All of the functions in the class could be shown toward the bottom of the palette, and there could be a filter box in order to help the programmer quickly find the function they need. This could also aid in discovering related functions to the programmer’s initial options that would be a good fit for program creation. This version maintains the important tools for “Example Code” and “Blocks of In-Scope Variables” as well as other common tools before the functions in the class.

For the scenario in figure 4.21, we took the previous example and added an error to it. In this example, the programmer needs to add “Library.Containers.Array” to the use statements at the top of the program in order to use the array class. The alert box promptly opens up when the cursor is present on the error line, but when the cursor is not on that line, the alert box will hide itself again. Clicking on the “Click Here to Add” link in the alert box will add the proper “use statement” to the top of the file and the alert box will disappear, effectively reverting the palette to look like figure 4.24.

Notably, each of these examples do not all necessarily contain the same tools nor the same number of tools. This is a feature and the power of the context-aware palette; it can adapt to a given scenario. Since programming is such a complex task, the scenarios above do not cover all possibilities. These provided scenarios are a general starting point for guidance. For example, it is easy to imagine any number of

Figure 4.21: Recommended Palette: Empty File

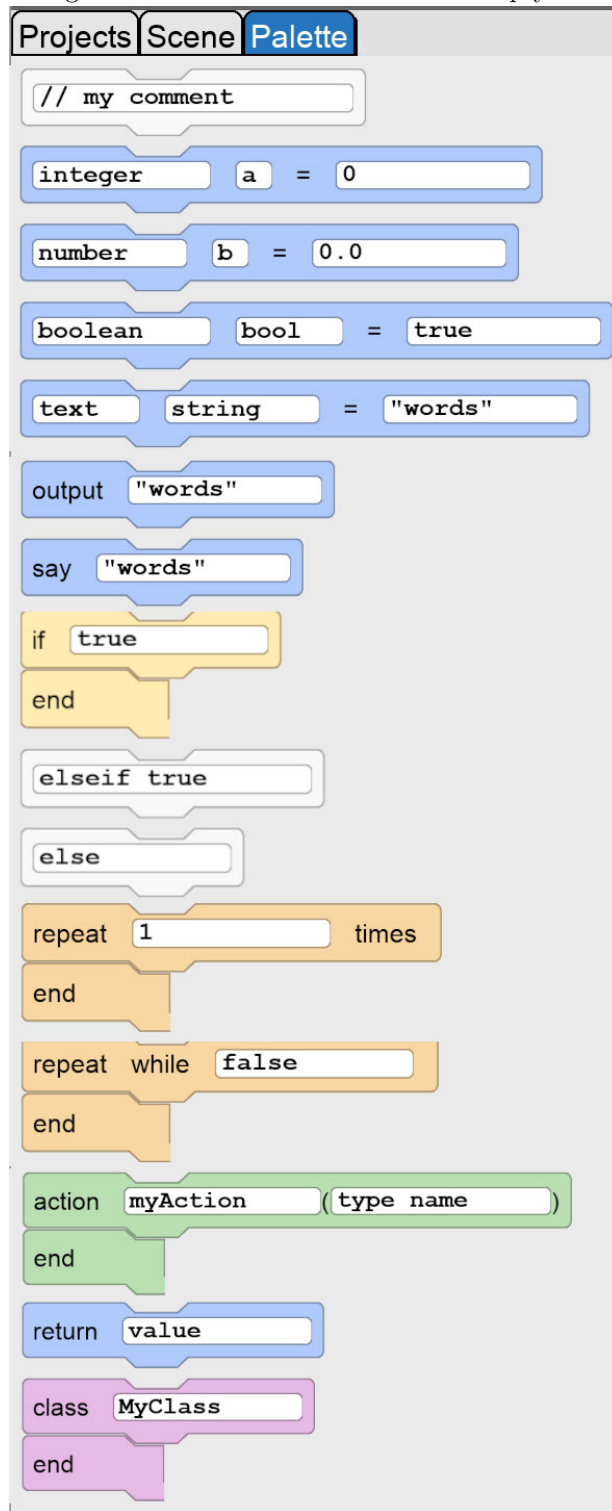


Figure 4.22: Recommended Palette: Editing a Script Without Imported Libraries

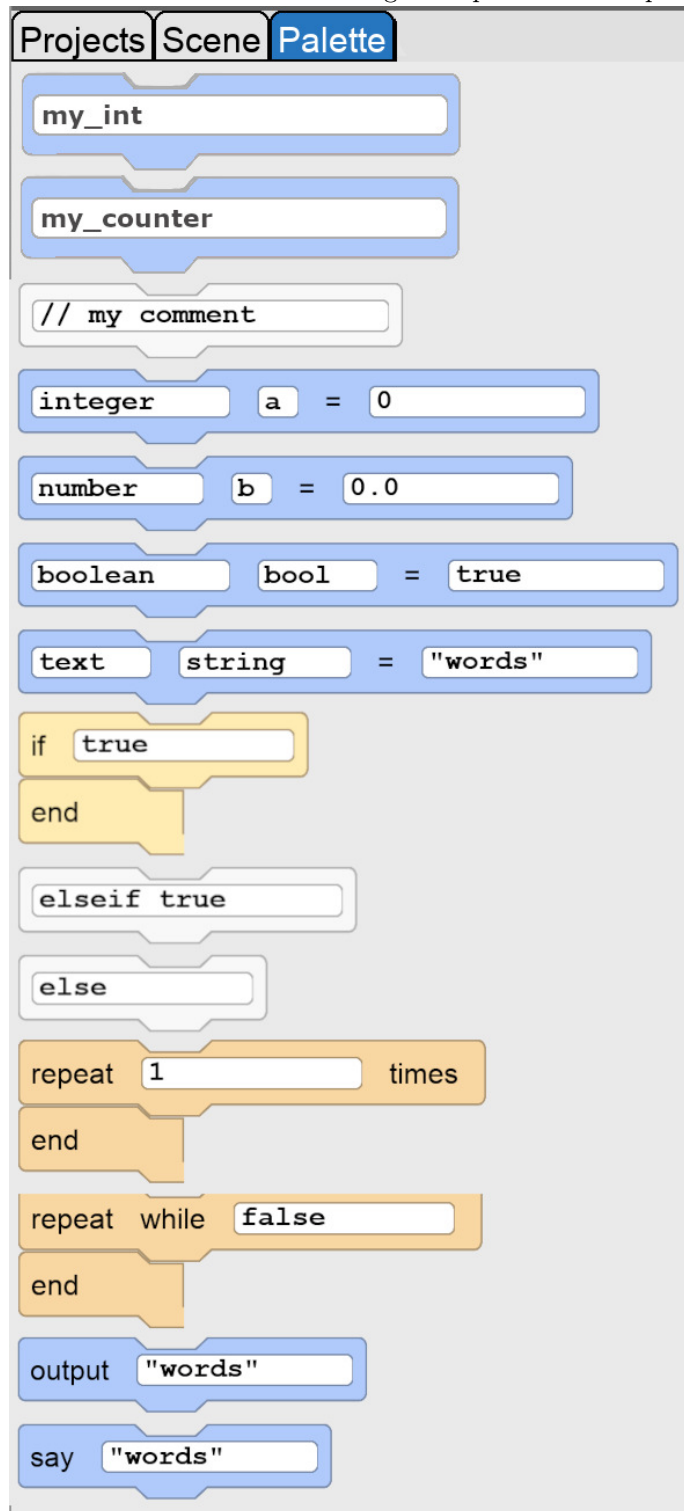


Figure 4.23: Recommended Palette: Editing a Script With Imported Libraries

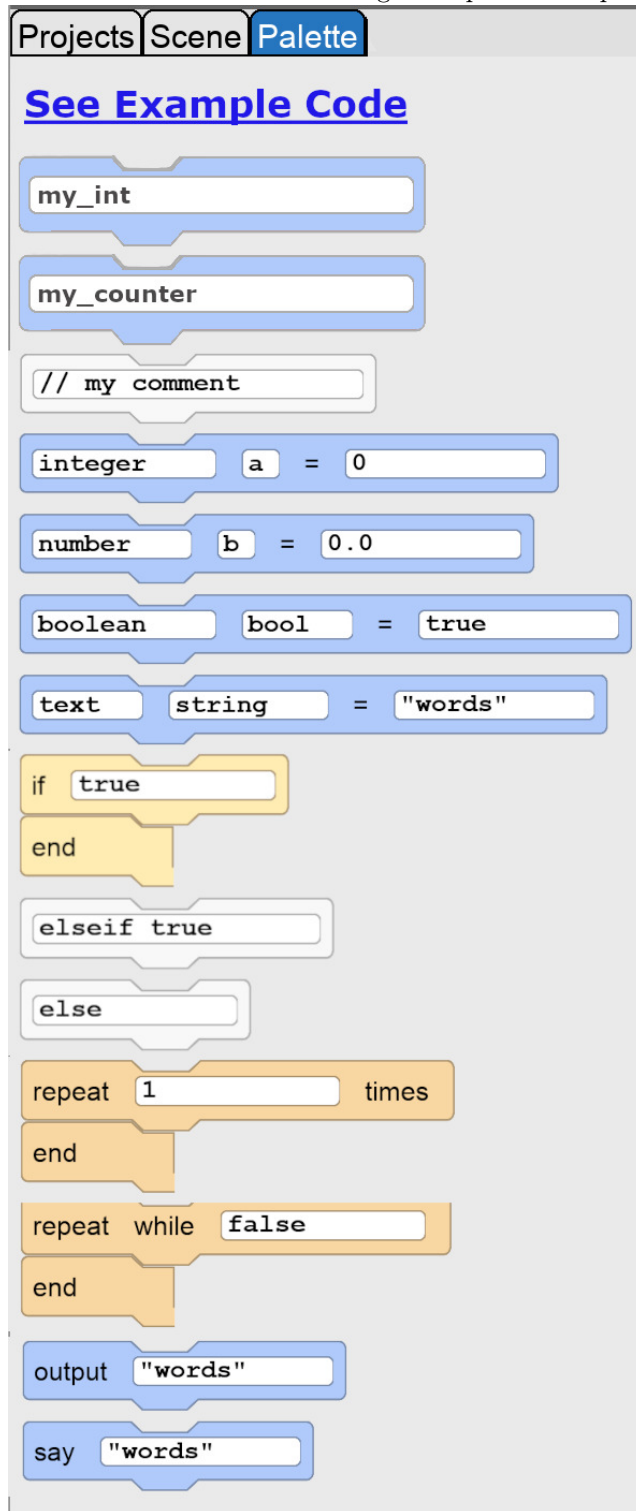


Figure 4.24: Recommended Palette: Editing a Class

The image shows a software development environment's 'Recommended Palette' for editing a class. At the top, there are three tabs: 'Projects', 'Scene', and 'Palette', with 'Palette' being the active tab. Below the tabs is a blue link that says 'See Example Code'. The palette contains several code snippets, each in a light blue rounded rectangle with a white background and a blue border. The snippets are: 'my\_int', 'my\_counter', '// my comment', 'integer a = 0', 'number b = 0.0', 'boolean bool = true', and 'text string = "words"'. Below these is a 'Filter' input field. Underneath the filter is a list of method call snippets, each in a similar rounded rectangle: 'Activate(event)', 'Activate()', 'Add(gutter)', 'Add(index, newItem)', 'Add(newItem)', 'AddBehavior(behavior)', 'AddBlockChangeListener(listener)', and 'AddControlActivationListener(listen'.



Figure 4.25: Recommended Palette: Error in the Code

The image shows a software interface with three tabs: 'Projects', 'Scene', and 'Palette'. The 'Palette' tab is active and contains an error message box with a red border. The error message reads: 'Import Missing!', 'Add use for: Library.Containers.Array', and a blue link 'Click Here to Add'. Below the error message is a blue link 'See Example Code'. Underneath are several code blocks, each with a blue border and a white background. The blocks contain the following code: 'my\_int', 'my\_counter', '// my comment', 'integer a = 0', 'number b = 0.0', 'boolean bool = true', and 'text string = "words"'. Below these blocks is a 'Filter' input field. At the bottom, there is a list of method calls: 'Activate(event)', 'Activate()', 'Add(gutter)', 'Add(index, newItem)', and 'Add(newItem)'. The interface has a light gray background.

scenarios similar to example two, but with specific libraries such as DataFrame or Array imported. In these cases, it might be beneficial to add “Blocks of All Functions in the Class” instead of “Blocks to Create New Blocks” to the palette. Overall, these three examples are to serve as a starting point rather than a final authoritative word for all use cases.

#### 4.4.7 Generalizability

This study encompassed participants ranging from early-stage novices to professionals with decades of experience. All participants completed high-school at a minimum, and some had degrees as high as a PhD. Professionals were those who are currently employed in programming roles or who had been in the past where programming was either the primary or secondary responsibility of the role. Despite these factors, the generalizability of this study is difficult to quantify. We did not get enough professional programmers to claim any level of generalizability for that group with this result. This study also excludes K–12 students, which is an important missing cohort for any potential generalizability.

Further, studies involving surveys have natural limitations. Due to the lack of a working tool to actually test, this study’s format as a thought experiment reduced the ability to effectively generalize our results. Since no validated instruments appear to exist, nor does such an IDE exist to create said instrument, the present study’s value should be seen as an initial stake in the ground from which to conduct future studies. We do not know if these results would hold when such a tool is used in practice; we would need more real-world observations, along with taking timing and accuracy measurements, with people across a spectrum of different skill levels to claim any level of generalizability. Put simply, the data from this study can be used to create an initial IDE for testing purposes; future studies can use that initial IDE to validate or refute the data herein.

#### 4.4.8 Implications

Similar to the first study, future research should be conducted in various ways. First, this study could be conducted on a larger sample, especially with more professional programmers. This would help generalize those results to a broader range of programming experiences. It is likely that utilizing a more sophisticated testing environment that allows for live programming would help test the palette once implemented. Finally, testing this instrument on K–12 novices would increase the generalizability of these findings. As before, it makes sense to consider “birth to death” as a metaphor to understand when and where such results might

apply.

#### 4.4.9 Limitations

A primary limitation of the study is the participant pool’s limited number of professionals. In order to partially compensate, we classified some upper-level students with professional programming experience as professionals, but since this is self-reported data, we do not know if this experience would typically be classified as professional experience or more like a basic internship. To compound that, the ongoing work of Siegmund along with Peitek et al. [155] demonstrate that self-reported programmer experience is not the primary factor in significantly different programmer efficacy. Instead, it is part of a multi-dimensional set of factors including interest in learning, self comparison/reflection, and others, so just selecting years of experience and professional experience have little to no power to predict outcomes.

Another limiting factor is the lack of participants who are K–12 students. It is impossible to say if such a tool can scale from early novices to professionals if we cannot start at the earliest stages. This cohort should be studied once an initial tool is made, so their results can be directly observed.

Another limitation is that this was a thought experiment for participants, since the instrument did not allow for any interaction nor programming activities. Participants might have different responses once such a system is implemented and they are able to interact with it. This was merely a thought experiment wrapped in a survey, but once participants can use it in practice, they may help us uncover completely different results.

Finally, and potentially the greatest limitation, there likely was not a broad enough array of scenarios tested. For example, “Blocks for Control” and “Blocks for Exiting” were ranked quite low in this study, but there were not any scenarios that specifically called for them. The results might be different if we added scenarios requiring loops or return statements. This is a difficult limitation to overcome in this format, since we would never be able to test every possible programming scenario. Thus the format itself was a major limitation, but it was beneficial for creating the first prototype.

# Chapter 5

## Conclusion

### 5.1 Discussion of Research Questions

Overall, this body of work endeavored to improve the ability for students to learn programming; each study approached this goal from a different aspect. We included input from professional programmers because they bring a wealth of knowledge and experience to programming overall. In addition, it was important to us to get feedback from university students, since they represent an array of experience levels. Some student participants had very introductory, cursory experience with programming, providing feedback from early-stage learners. Other student participants represented an intermediate level of programming experience. Each of the studies in chapter 3 and chapter 4 were designed to broadly address the initial research questions, which we will now discuss in turn.

#### 5.1.1 RQ 1

*What attributes of block programming languages and environments do professionals find to be beneficial versus detrimental?*

In this question, we assessed what existing attributes of block-based languages and environments help programmers succeed in learning and coding, and conversely, which ones are at least potentially detrimental to learning and learning transference.

In the chapter 3 study about block-based programming attributes, we identified several attributes of block languages that are beneficial to programming and learning programming. As previously noted in studies, colorful blocks are beneficial despite the fact that we do not empirically know the reason why, nor

what about them makes them beneficial. The shape of the blocks and the dropdown arrows are beneficial to learning and programming, since they provide visual markers to easily distinguish information. Perhaps the most obvious beneficial attribute is one we labeled as “text, punctuation, and symbols,” which simply provides the same information in a block language that it would in a text-based language.

When we consider the chapter 4 study about the IDE for block languages, we find that several of the basic blocks, which most block-based IDEs provide by default, are helpful for programming activities. In this study, we were not surprised to learn it is beneficial for the IDE to present blocks for controls, conditionals, input/output, and creating new variables.

Conversely, there were several attributes working in concert to form spatial layout, that appear to be detrimental to the learning and programming environment. Spatial layouts, which are dispersed on an unbound editing palette, negatively impact programming activities for several reasons. First, horizontal scrolling and dispersed block arrangement can be taken as a problematic pair. It is a much more common and simple activity to scroll up and down rather than left and right. This is even more true when accessibility is considered, which it should always be. There are more options to easily scroll up and down to suit people’s needs. For example, up and down arrows can be used to simply move line by line vertically. Moving left and right with the keyboard typically moves within a line rather than from left entity to right entity, such as from a left set of blocks to a different set of blocks on the right. Using the tab key does not solve that complication, since tab is used for indentation or other special uses in accessible environments. Thus, a combination of keystrokes would be needed to effectively move between left/right positioned blocks, which may present other accessibility challenges, such as for people with mobility issues. Additionally, spatial layout prevents the ability to use line numbers, for which programmers indicated a preference. Thus, we highly recommend against utilizing spatial layouts.

### 5.1.2 RQ 2

*What aspects of block-based programming might be beneficial to both learners and professionals?*

The intent behind this question was to determine if there is a difference between the perspectives of professionals and learners. In many cases, such as throughout the study in chapter 4 about the IDE tools, we learned there was not a significant difference between the perspectives of professionals and novices.

In the chapter 3 study about visual attributes, we discovered significant differences between learners and professionals for the attributes of color, text, and line numbers. We also found differences, as expected, with task timing and correctness. In all of these cases, while the differences were statistically significant,

the effect size was very small. When testing the color attribute, professionals in the two groups which received the horizontal scroll treatment were more extreme in rating the preference for color or the disdain for grayscale; that experience difference accounted for 6% of the model, which is fairly small. In all other cases of professionals versus learners perspectives, the differences were even smaller. They were not more than 2.5% and ranged down to less than 1%.

Taking this all into account, while there were some differences between professionals and learners, in this study we did not observe large differences between the two experience groups. Perhaps the differences would be greater if live programming tasks were possible, but that might also present greater negative effects with programming anxiety [156]. The differences might also present as stronger effect sizes if the tasks were more difficult in nature; the simplicity of tasks made them more achievable by student novices, thus, the differences may not have been as great. These results would best be tested with an online system where live programming activities can take place, and the system should automatically log the time to correct completion without letting participants move to the next task unless they either (a) correctly solve the challenge or (b) hit a pre-determined time limit per task. Additionally, it should be considered if the task difficulty should be static or build in difficulty, and at what difficulty each level of programming task should be. With such changes and considerations, results could be further evaluated to determine if this research question is further supported or refuted in subsequent studies.

### 5.1.3 RQ 3

*What capabilities could be added to a block language and environment to ease the transitional friction for learners to become professionals?*

Here, the purpose was to find if there are gaps in current block languages that could help with learning to program, transference of that learning, and programming in block languages in general. If done properly, the need for a transfer of learning might be diminished because the block language and environment would be robust enough to be suitable for a broader set of programming purposes, which means more programmers would be able to use it for more tasks during a longer tenure.

The first beneficial addition to block languages is to have a single, left-aligned column of blocks, at least for left-to-right language readers. With the aforementioned discussion about spatial layout, we recognized the inability to have line numbers, despite programmers' preference for them. Employing this layout also prevents the dispersed layout and the need for wide-scale horizontal scrolling. It would also prevent times when spatial layout with blocks results in some blocks getting orphaned on the editing palette

or the case of not knowing which block groups will run first. Having the single, left-aligned column of blocks prevents many of the detrimental issues empirically assessed in the two studies in this dissertation. Programmers would instead be able to rely upon vertical scrolling, consistent layout, line numbers, reliable parse/execution order, and not losing any of their code to an unfindable location.

Another potential beneficial addition to block languages is the ability to create comments. While there are some block languages that allow for comments, many of them hide the comments inside of a popup or another area that is difficult to discover. Comments serve various purposes and should be added to programs to aid with rapid program comprehension [157, 158], which is critical, considering that prior studies demonstrate that program comprehension takes over half the time spent in development activities [132, 133]. Providing the ability to add comments prominently in special-made blocks for comments, rather than hiding them or disallowing them, will help learners to better understand and professionals to more quickly comprehend preexisting code.

Creating a context-aware palette for a block-based IDE is another potential way to make block languages beneficially scale from learners to professionals. There are multiple tools, or features/affordances, that can be added to a context-aware palette beyond just the standard set of blocks outlined in the discussion of RQ 1. We will address them here in order of preference from the second study in chapter 4 about the context-aware IDE.

First, example code was the most commonly-requested tool for a context-aware palette. While initially this might seem to indicate that example code should always be present, in fact it is most powerful when used in specific contexts. For example, attempting to display example code in an empty or near-empty file is not beneficial, and in fact it could get in the way of more useful tools for that stage of code implementation. Similarly, there are times in writing programs, such as a simple “if statement” in a script, where example code may cause cognitive overload rather than serve as a helpful tool. Where example code could truly display its power is when the cursor is on specific blocks, such as classes or functions/methods; thus, the palette could provide access to example code about the usage of those blocks. This advanced feature would be quite powerful for intermediate and advanced programmers who are learning new programming language features.

Another context-aware feature that would benefit programmers from novices through professionals is blocks of in-scope variables. Many block languages are able to add user-created variable blocks to their existing palette. The power behind a context-aware palette presenting in-scope variables is two-fold. For one, the palette would not present blocks outside of the scope where the cursor sits, so there would be

reduced clutter on the palette. Secondly, the context-aware palette could show in-scope variables that were inherited from other classes in the scope. These would both be beneficial as a learning tool about scope, as well as a means of enhancing efficiency for advanced programmers.

Alerts about problems and how to fix them were highly ranked across the board by programmers. This tool would provide the capability to see in real time when an error existed. Current IDEs often show a small red dot in the margin of the code indicating when an error exists. In the example code we provided in the study from chapter 4 about the IDE, the entire block turned red to indicate there was an issue. If, in addition to this red alert color, a new section of the context-aware palette appeared and displayed a plainly worded explanation of the problem, the programmer would be able to easily fix it. Additionally, if the error is as simple as needing to add a new “use statement,” as in the case of our example in the study, the error explanation section could have a button for the programmer to click that would automatically add said “use statement” to the top of the program. Having effective and prominent messaging for syntax errors is critical, especially for novices [159, 160, 89].

The final two suggested additions will be addressed together: blocks for all the functions in a given class and links to class documentation. Having the capability to show these options would be a beneficial learning tool for novices as well as a shortcut and helper tool for advanced programmers. It would be entirely possible to show both tools, but it may be redundant to do so. In the case of showing blocks for all functions in the class, having a method to filter, such as a text box to type in part of the name, would provide a shortcut to access the needed blocks. In addition, this method would allow discoverability to lesser-used functions, especially for advanced programmers. For example, it would be easy to imagine an array class, such as in the Quorum programming language, having a function to add an element to an array along with the value and location, like this: “Add(integer location, Type value)” in a given language. If the programmer typed “Add” into the filter box, they may discover more helpful functions such as “AddToEnd(Type value)” and “AddToFront(Type value)” that would automatically handle adding the value at the location for them. Providing the functionality to automatically generate all the blocks in a given class, along with having a filter box, could speed up professional programmers’ work as well as be a tool for novices to expand their programming knowledge. Similarly, having links to class documentation would be beneficial in that the programmer would have easy, one-click access to all the documentation, which would provide all the information about what classes are available and how to use them without any arduous searching. It could additionally have example code, which is the most requested tool addition to a context-aware palette.



Taken all together, these capabilities could enhance the capability for novices to learn programming, the efficiency of professional programmers, and potentially reduce the transitional friction from blocks to text by providing a robust block-based programming language and environment to keep programmers using blocks from the learning stage through professional programming.

## 5.2 Future Research

Numerous studies could expand on this study of block-based attributes. The study could be replicated with live programming tasks and with some minor instrumentation changes, such as timing, forcing correctness within a time limit, and the ability to vary treatments, to see if the findings from this present study hold. Such a system, and its online availability, would provide programmers with the ability to actually solve live coding challenges. The first study in chapter 3 about visual attributes in a block-based language could be fully replicated within such an environment to determine if the results hold true.

Another interesting independent study to pursue is one on color usage. Much is stated about block languages using colors as a memory and programming aid, yet many studies lament the low transference of learning from blocks to text. There is a discordance, in this researcher's opinion, between how colors are treated in block languages, which are done by functional groups, versus in text-based languages, which are colored by syntactic groups. It is entirely possible that some of the transference of learning issues are due to the aforementioned discordance, so a study should be performed to test that idea.

There are several studies that could be performed after the initial creation of a context-aware IDE. First, the second study in chapter 4 could be fully replicated as-is with live coding challenges. As stated above, with some development work, the study could be replicated with live programming tasks to see if the findings hold. Additional studies could be done with varying treatments, such as changed tool orders or a control group using static, non-context-aware tools for comparison. With those studies, task timing and accuracy could come into play and be better measures than user-selected preferences.

In both cases, additional studies could be repeated with more professional programmers. Perhaps even more importantly, the studies could be repeated with K–12 students and teachers. Block-based programming is extremely popular for early-stage computer science education, so input from K–12 students and teachers is critical to ensuring we are serving this important group. Of particular interest is K–4, in which students are in the early stages of learning how to read, so a block language needs to be usable for pre-literate students. This would include a transitional age range where students could use a programming

language without words and proceed into a programming language with simple words. Following that, there is the next transitional state from simplified words to more complex, expressive language as students transition to high school. Throughout these age ranges, there are multiple states where transitional friction may occur, so any studies that expand on these works would be beneficial to computer science education and computational thinking overall.

# Appendix A

## Survey Instrument 1

The following is the instrument that was created to assess the attributes in chapter 3, Study 1: Assessing Visual Attributes' Role in Readability and Comprehension of Block-Based Programs.

Tools ▾

Saved at 4:43 PM Draft



Preview

Publish

## block-based-programming-irb-req

▾ Consent ⋮

Consent

## Block-Based Programming

Howard Hughes School of Engineering

Computer Science

UNLV

### INFORMED CONSENT FORM:

Before you consider the research, you should be aware of the following information:

- Research is voluntary. You do not have to be in this research study.
- There are no risks to you from participating in this research study except for your time and inconvenience.
- You will fill out a survey with basic demographic information about yourself.
- You will work through some programming tasks and answer questions about them.
- The research study will take approximately 15 minutes.
- If you agree to be in the study, you should read the rest of this document. The document explains what will happen to people in the study.
- You must be at least 18 years of age to participate.

#### WHY ARE YOU BEING ASKED TO PARTICIPATE:

The purpose of the research is to learn about how block-based programming languages are used by practitioners.

#### WHAT YOU WILL BE ASKED TO DO:

If you decide to participate, you will be asked to fill out a demographic survey (e.g., age, gender, etc), work through some problems related to reading software programs, and provide feedback.

#### RISKS:

Except for your time and inconvenience, there are no risks to you from participating in this study.

#### BENEFITS:

While this study will have no direct benefit to you, this research may help you learn more about programming and programming environments.

#### CONFIDENTIALITY:

Your name will not be on any of the data. Your name or other identifying information will not be reported in any publications. The de-identified data could be used for future research studies or distributed to another investigator for future research studies without additional informed consent from you.

#### VOLUNTARY:

Participation is voluntary. If you choose to take part in this study, you may stop at any time.

#### CONTACT INFORMATION:

If you have any questions about this study, please contact Alex Hoffman at alex.hoffman@unlv.edu or Dr. Andreas Stefik at andreas.stefik@unlv.edu. If you have any questions about your rights as a research participant, please contact the Office of Research Integrity, University of Nevada Las Vegas, at ORI@unlv.edu.

If you click Accept, it indicates that you have read the above information and agree

Consent Answer \*

Do you consent to be in this study?

Yes, I Accept

No, I Do Not Accept

[Import from library](#) [Add new question](#)

Demographics

D1 \*

What is the highest level of education you completed?

High School

Some College

Associate Degree

Bachelors Degree

Masters Degree

PhD

Other

D2 \*

If you are currently enrolled in college, what is your classification?

Freshman

Sophomore

Junior

Senior

Graduate

Not Enrolled

D3 💡 \*

How old are you?

D4 \*

What is your gender?

- Male
- Female
- Non-binary / third gender
- Prefer not to say

D5 \*

What is your ethnicity?

- White
- Black or African American
- American Indian or Alaska Native
- Asian
- Native Hawaiian or Pacific Islander
- Other
- Prefer not to say

D6 \*

Are you now or have you ever been employed to write code professionally, regardless of the programming language?

- Currently employed to write code
- Previously employed to write code
- Never employed to write code as a primary job function, but it was a periodic part of the job
- Never employed to write code

D7 \*

What programming languages have you used to write code?

- C / C++
- Java
- Javascript
- HTML / CSS
- PHP
- Python
- Other
- 
- None

D8 💡 ✖

How many years of programming experience do you have?

[Import from library](#) [Add new question](#)


Explanation1\_clr

E1

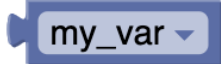
You will be introduced to a block-based programming language. You will then be asked a series of questions about the block-code that is presented to you.

Variables are used to store information to be referenced and manipulated in a program.

This block **creates** or initializes a variable (named `my_var`).



This block **uses** an existing variable (named `my_var`).



[Import from library](#) [Add new question](#)

Explanation2\_clr



E2

Here are some basic types of data.

This is an integer, which is a whole number:



This is how to set the variable my\_var to the integer 5:



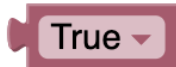
This is a string, which is text-based data that can contain letters, numbers, spaces, and special characters:



This is how to set the variable my\_var to the string "Hello world!":



This is a bool, which can only be either True or False:



This is how to set the variable my\_var to True:



[Import from library](#)

[Add new question](#)

▼ T1-Q1a-GrpA

T1a

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange(1, 100)
set guess = 0
set guesses = create empty list []
define check_num
  parameters:
    num
    guess
  if guess > num
    return "high"
```

T1a-1



How many times is a variable **created** in the code above? (if any are in a loop, count it as 1)

Type the number below or "unsure" if you are unclear or need more info.

Q15

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

▼ T1-F1a\_GrpA

Q291

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange(1, 100)
set guess = 0
set guesses = create empty list []
define check_num
  parameters:
    num
    guess
  if guess > num
    return "high"
```

T1F1a 💡 ✖

For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

▲ 📄 Import from library ➕ Add new question

▼ T1-Q1b-GrpA

T1b

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"  
import random  
set num = random.randrange(1, 100)  
set guess = 0  
set guesses = create empty list []  
define check_num  
  parameters:  
    num  
    guess  
  if guess > num  
    return "high"
```

T1b-1



How many times is a variable **used** (not set) in the code above?  
Note: it does not have to be executed to be used. Any use in a loop counts as 1 time.

Type the number below or "unsure" if you are unclear or need more info.

Q25

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

▼ T1-F1b\_GrpA

Q315

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"  
import random  
set num = random.randrange ( 1 100 )  
set guess = 0  
set guesses = create empty list []  
define check_num  
  parameters:  
    num  
    guess  
  if guess > num  
    return "high"
```

Q316



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1c-GrpA

T1c

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"  
import random  
set num = random.randrange ( 1 100 )  
set guess = 0  
set guesses = create empty list []  
define check_num  
  parameters:  
    num  
    guess  
  if guess > num  
    return "high"
```

T1c-1



How many strings are in the code above?

Note: Count only strings inside of quotation marks. Count each string inside of quotation marks as one, even if it is within a loop. Count each string separately, even if they are on the same line.

Type the number below or "unsure" if you are unclear or need more info.

Q21

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#) [Add new question](#)

▼ T1-F1c\_GrpA



Q317

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange(1, 100)
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num, guess
  if guess > num
    return "high"
```

Q318



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1d-GrpA

T1d

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"  
import random  
set num = random.randrange(1, 100)  
set guess = 0  
set guesses = create empty list []  
define check_num  
  parameters:  
    num  
    guess  
  if guess > num  
    return "high"
```

T1d-1



Which variable was **used** the most? (not set)

Note: Count each usage inside as one. Assume any loops only execute one time.

- ERR\_MSG
- num
- guess
- guesses
- result
- Unsure or need more information

Q184

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1d\_GrpA

Q319

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "  
import random  
set num = random .randrange ( 1 100 )  
set guess = 0  
set guesses = create empty list []  
define check_num  
  parameters:  
    num  
    guess  
  if guess > num  
    return " high "
```

Q320 💡 ✖

For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1a-GrpB

T1Q1a

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"
```

T1q1-a



How many times is a variable **created** in the code above? (if any are in a loop, count it as 1)

Type the number below or "unsure" if you are unclear or need more info.

Q190

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#)

[Add new question](#)

T1-F1a\_GrpB

Q328

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " corr
```

Q329



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1b-GrpB



T1Q1b

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"
```

T1Q1-b



How many times is a variable **used** (not set) in the code above?  
Note: it does not have to be executed to be used. Any use in a loop counts as 1 time.

Type the number below or "unsure" if you are unclear or need more info.

Q200

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#) [Add new question](#)

T1-F1b\_GrpB

Q330

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " corr
```

Q331



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

▼ T1-Q1c-GrpB

T1Q1c

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"
```

T1q1c-1



How many strings are in the code above?

Note: Count only strings inside of quotation marks. Count each string inside of quotation marks as one, even if it is within a loop. Count each string separately, even if they are on the same line.

Q206

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#)

[Add new question](#)

T1-F1c\_GrpB

Q332

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " corr
```

Q333



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Helpful nor Harmful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1d-GrpB

T1Q1d

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange(1, 100)
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess > num:
return "high"
elif guess < num:
return "low"
else:
return "correct"
```

T1Q1d-1



Which variable was **used** the most? (not set)

Note: Count each usage inside as one, even if it is within a loop.

- ERR\_MSG
- num
- guess
- guesses
- result
- Unsure or need more information

Q209

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1d\_GrpB

Q334

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"
```



Q335



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

Explanation3\_clr

E3

Here is how to print data to the program's user.

This is the print block, which prints the item inside parentheses () to the user

```
print ( )
```

This is how to print the variable my\_var:

```
print ( my_var )
```

This is how to print the string "Hello world!":

```
print ( " Hello world! " )
```

This is how to print an integer:

```
print ( 5 )
```



[Import from library](#) [Add new question](#)

Explanation4\_clr

Q342

Here is how to receive input from the program's user.

This is the input block, which prints any text inside quotation marks to the user:

```
input ( " " )
```

Here, the question "What is your name" is printed to the user, and the program waits for the user to type in an answer and press return:

```
input ( " What is your name? " )
```

This group of blocks will print the text "What is your name" to the user and wait for a response. When the user responds, the data they entered will be stored to the variable my\_var:

```
set my_var = input ( " What is your name? " )
```

[Import from library](#)

[Add new question](#)

▼ T1-Q1e-GrpA

T1e

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange(1, 100)
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num, guess
  if guess > num
    return "high"
```

T1e-1

\*

What is the **first** line of text displayed when the user runs the program?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information

Q29

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1e\_GrpA

Q321

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num guess
  if guess > num
    return " high "
```

Q322



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1f-GrpA

T1f

Use this code to answer the questions below. You might need to scroll down to see all the code.

```

set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters:
    num
    guess
  if guess > num
    return "high"

```

T1f-1

\*

Assume num is set to 95 and the user enters the integer 5. What is the **next single line of text displayed** to the user?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information

Q33

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1f\_GrpA

Q323

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"  
import random  
set num = random.randrange(1, 100)  
set guess = 0  
set guesses = create empty list []  
define check_num  
  parameters:  
    num  
    guess  
  if guess > num  
    return "high"
```



Q324



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1g-GrpA

T1g

Use this code to answer the questions below. You might need to scroll down to see all the code.

```

set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters:
    num
    guess
  if guess > num
    return "high"

```

T1g-1

\*

Assume num is set to 50 and the user enters "help" at the prompt. What is the **next single line of text displayed** to the user?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more info

Q38

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1g\_GrpA

Q325

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"  
import random  
set num = random.randrange(1, 100)  
set guess = 0  
set guesses = create empty list []  
define check_num  
  parameters:  
    num  
    guess  
  if guess > num  
    return "high"
```

Q326 💡 ✖

For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

▲ 📄 Import from library ➕ Add new question

▼ T1-Q1e-GrpB

T1Q1e

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

def check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"
```

T1Q1e-1



What is the **first** line of text displayed when the user runs the program?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information

Q212

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1e\_GrpB

Q336

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " corr
```

Q337



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1f-GrpB

T1Q1f

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```

set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"

```

T1Q1f-1



Assume num is set to 95 and the user enters the integer 5. What is the **next single line of text displayed** to the user?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information



Q215

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1f\_GrpB

Q338

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"
```

Q339



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1g-GrpB

T1Q1g

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```

set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess > num
return "high"
elif guess < num
return "low"
else:
return "correct"

```

T1Q1g-1



Assume num is set to 50 and the user enters "help" at the prompt. What is the **next single line of text displayed** to the user?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information

Q218

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1g\_GrpB

Q340

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
```

```
define check
param
if
return "
elif
return "
else:
return "
```

Q341



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

Explanation1\_bw

Q344

You will be introduced to a block-based programming language. You will then be asked a series of questions about the block-code that is presented to you.


Variables are used to store information to be referenced and manipulated in a program.

This block **creates** or initializes a variable (named my\_var).



This block **uses** an existing variable (named my\_var).



 Import from library

Add new question

▼ Explanation2\_bw

Q346

Here are some basic types of data.

This is an integer, which is a whole number:



This is how to set the variable my\_var to the integer 5:



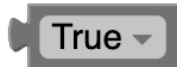
This is a string, which is text-based data that can contain letters, numbers, spaces, and special characters:



This is how to set the variable my\_var to the string "Hello world!":



This is a bool, which can only be either True or False:



This is how to set the variable my\_var to True:



[Import from library](#)

[Add new question](#)

▼ T1-Q1a-GrpC

Q347

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num guess
  if guess > num
    return " high "
  elif guess < num
```

Q348



How many times is a variable **created** in the code above? (if any are in a loop, count it as 1)

Type the number below or "unsure" if you are unclear or need more info.

Q349

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.



[Import from library](#) [Add new question](#)

▼ T1-F1a\_GrpC



Q350

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters:
    num
    guess
  if
    guess > num
    return " high "
  elif
    guess < num
```

Q351 💡 ✖

For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

▲ 📄 Import from library ➕ Add new question

▼ T1-Q1b-GrpC

Q352

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "  
import random  
set num = random .randrange ( 1 100 )  
set guess = 0  
set guesses = create empty list []  
define check_num  
  parameters:  
    num  
    guess  
  if guess > num  
    return " high "  
  elif guess < num
```

Q353



How many times is a variable **used** (not set) in the code above?  
Note: it does not have to be executed to be used. Any use in a loop counts as 1 time.

Type the number below or "unsure" if you are unclear or need more info.

Q354

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#)

[Add new question](#)

▼ T1-F1b\_GrpC

Q413

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "  
import random  
set num = random .randrange ( 1 100 )  
set guess = 0  
set guesses = create empty list []  
define check_num  
  parameters:  
    num  
    guess  
  if guess > num  
    return " high "  
  elif guess < num
```

Q414



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

▼ T1-Q1c-GrpC

Q355

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num guess
  if guess > num
    return " high "
  elif guess < num
```

Q356



How many strings are in the code above?

Note: Count only strings inside of quotation marks. Count each string inside of quotation marks as one, even if it is within a loop. Count each string separately, even if they are on the same line.

Type the number below or "unsure" if you are unclear or need more info.

Q357

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#) [Add new question](#)

▼ T1-F1c\_GrpC

Q415

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters:
    num
    guess
  if
    guess > num
    return " high "
  elif
    guess < num
```

Q416



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Helpful nor Harmful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1d-GrpC



Q358

Use this code to answer the questions below. You might need to scroll down to see all the code.

```

set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num
             guess
  if guess > num
    return " high "
  elif guess < num

```

Q359



Which variable was **used** the most? (not set)

Note: Count each usage inside as one. Assume any loops only execute one time.

- ERR\_MSG
- num
- guess
- guesses
- result
- Unsure or need more information

Q360

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#)
[Add new question](#)

T1-F1d\_GrpC

Q417

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num
             guess
  if guess > num
    return " high "
  elif guess < num
```

Q418



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

▼ T1-Q1a-GrpD

Q361

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " corr
```

Q362



How many times is a variable **created** in the code above? (if any are in a loop, count it as 1)

Type the number below or "unsure" if you are unclear or need more info.

Q363

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#) [Add new question](#)

T1-F1a\_GrpD

Q393

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
```

```
import random
```

```
set num = random .randrange ( 1 100 )
```

```
set guess = 0
```

```
set guesses = create empty list []
```

```
define check_num
```

```
  parameters:
```

```
  if
```

```
    guess
```

```
  return " high
```

```
  elif
```

```
    guess
```

```
  return " low
```

```
  else:
```

```
  return " cor
```

Q394



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

▼ T1-Q1b-GrpD

Q364

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " corr
```

Q365



How many times is a variable **used** (not set) in the code above?  
Note: it does not have to be executed to be used. Any use in a loop counts as 1 time.

Type the number below or "unsure" if you are unclear or need more info.

Q366

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#) [Add new question](#)

T1-F1b\_GrpD

Q425

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
```

```
import random
```

```
set num = random .randrange ( 1 100 )
```

```
set guess = 0
```

```
set guesses = create empty list []
```

```
define check_num
```

```
parameters:
```

```
if guess
```

```
return " high
```

```
elif guess
```

```
return " low
```

```
else:
```

```
return " cor
```



Q426



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Helpful nor Harmful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

▼ T1-Q1c-GrpD

Q367

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " corr
```

Q368



How many strings are in the code above?

Note: Count only strings inside of quotation marks. Count each string inside of quotation marks as one, even if it is within a loop. Count each string separately, even if they are on the same line.

Q369

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#)

[Add new question](#)

T1-F1c\_GrpD

Q427

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
```

```
import random
```

```
set num = random .randrange ( 1 100 )
```

```
set guess = 0
```

```
set guesses = create empty list []
```

```
define check_num
```

```
parameters:
```

```
if guess
```

```
return " high
```

```
elif guess
```

```
return " low
```

```
else:
```

```
return " cor
```

Q428



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Helpful nor Harmful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

▼ T1-Q1d-GrpD

Q370

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " corr
```

Q371



Which variable was **used** the most? (not set)  
Note: Count each usage inside as one, even if it is within a loop.

- ERR\_MSG
- num
- guess
- guesses
- result
- Unsure or need more information

Q372

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1d\_GrpD

Q429

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
```

```
import random
```

```
set num = random .randrange ( 1 100 )
```

```
set guess = 0
```

```
set guesses = create empty list []
```

```
def check_num
```

```
parameters:
```

```
if guess
```

```
return " high
```

```
elif guess
```

```
return " low
```

```
else:
```

```
return " cor
```

Q430



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

Explanation3\_bw

Q373

Here is how to print data to the program's user.

This is the print block, which prints the item inside parentheses () to the user

```
print ( )
```

This is how to print the variable my\_var:

```
print ( my_var )
```

This is how to print the string "Hello world!":

```
print ( " Hello world! " )
```

This is how to print an integer:

```
print ( 5 )
```



[Import from library](#)

[Add new question](#)

▼ Explanation4\_bw



Q374

Here is how to receive input from the program's user.

This is the input block, which prints any text inside quotation marks to the user:

```
input ( " " )
```

Here, the question "What is your name" is printed to the user, and the program waits for the user to type in an answer and press return:

```
input ( " What is your name? " )
```

This group of blocks will print the text "What is your name" to the user and wait for a response. When the user responds, the data they entered will be stored to the variable my\_var:

```
set my_var = input ( " What is your name? " )
```

[Import from library](#) [Add new question](#)

▼ T1-Q1e-GrpC

Q375

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num
             guess
  if guess > num
    return " high "
  elif guess < num
```

Q376



What is the **first** line of text displayed when the user runs the program?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information

Q377

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1e\_GrpC

Q419

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
```

```
import random
```

```
set num = random .randrange ( 1 100 )
```

```
set guess = 0
```

```
set guesses = create empty list []
```

```
define check_num
```

```
  parameters: num guess
```

```
  if guess > num
```

```
    return " high "
```

```
  elif guess < num
```

Q420 💡 ✖

For each of the following **visual indicators**, please select how **helpful or unhelpful** it was in answering the previous question.

	Extremely Unhelpful	Very Unhelpful	Somewhat Unhelpful	Neither helpful nor unhelpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1f-GrpC

Q378

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num guess
  if guess > num
    return " high "
  elif guess < num
```

Q379



Assume num is set to 95 and the user enters the integer 5. What is the **next single line of text displayed** to the user?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information

Q380

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1f\_GrpC

Q421

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num guess
  if guess > num
    return " high "
  elif guess < num
```

Q422



For each of the following **visual indicators**, please select how **helpful or unhelpful** it was in answering the previous question.

	Extremely Unhelpful	Very Unhelpful	Somewhat Unhelpful	Neither helpful nor unhelpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)



T1-Q1g-GrpC

Q381

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters:
    num
    guess
  if
    guess > num
    return " high "
  elif
    guess < num
```

Q382



Assume num is set to 50 and the user enters "help" at the prompt. What is the **next single line of text displayed** to the user?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more info



Q383

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

[Import from library](#)

[Add new question](#)

T1-F1g\_GrpC

Q423

Use this code to answer the questions below. You might need to scroll down to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []
define check_num
  parameters: num
             guess
  if guess > num
    return " high "
  elif guess < num
```

Q424



For each of the following **visual indicators**, please select how **helpful or unhelpful** it was in answering the previous question.

	Extremely Unhelpful	Very Unhelpful	Somewhat Unhelpful	Neither helpful nor unhelpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							

[Import from library](#) [Add new question](#)

▼ T1-Q1e-GrpD

Q384

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"
```

Q385



What is the **first** line of text displayed when the user runs the program?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information

Q386

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1e\_GrpD

Q431

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " cor
```

Q432



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Helpful nor Harmful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

▼ T1-Q1f-GrpD

Q387

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```

set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"

```

Q388



Assume num is set to 95 and the user enters the integer 5. What is the **next single line of text displayed** to the user?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information

Q389

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1f\_GrpD

Q433

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random .randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " cor
```

Q434



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely Harmful	Very Harmful	Somewhat Harmful	Neither Harmful nor Helpful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)

▼ T1-Q1g-GrpD



Q390

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = "Input an integer from 1 to 100"
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return "high"
elif guess
return "low"
else:
return "correct"
```

Q391



Assume num is set to 50 and the user enters "help" at the prompt. What is the **next single line of text displayed** to the user?

- Input an integer from 1 to 100
- high
- low
- correct
- Choose an integer from 1 to 100
- Your answer is high
- Your answer is low
- Your answer is correct
- It took
- guesses to guess correctly
- None: the program crashes
- Unsure or need more information

Q392

This question lets you record and manage how long a participant spends on this page. This question will not be displayed to the participant.

Import from library

Add new question

T1-F1g\_GrpD

Q435

Use this code to answer the questions below. You might need to scroll to the right to see all the code.

```
set ERR_MSG = " Input an integer from 1 to 100 "
import random
set num = random.randrange ( 1 100 )
set guess = 0
set guesses = create empty list []

define check_num
parameters:
if guess
return " high
elif guess
return " low
else:
return " cor
```

Q436



For each of the following **visual indicators**, please select how **helpful or harmful** it was in answering the previous question.

	Extremely harmful	Very harmful	Somewhat harmful	Neither helpful nor harmful	Somewhat Helpful	Very Helpful	Extremely Helpful
	1	2	3	4	5	6	7
Color of the block							
Shape of the block							
Arrangement of the blocks							
Scrolling							
Text, punctuation, or symbols							
Lack of comments							
Lack of line numbers							
Spacing							
Dropdown arrow next to text							
Other							



[Import from library](#) [Add new question](#)



Final

Q289 💡

Thank you for your time. If you have any additional feedback, please record it in the textbox below.

Q157 ★

If you wish to enter your email address to verify you took this survey, please select "Yes" to be directed to a new form. Your email address will not be associated with your survey responses.

Otherwise select "No" to end the survey without recording your email.

Yes

No

[Import from library](#) [Add new question](#)

[Add Block](#)

End of Survey

We thank you for your time spent taking this survey.

Your response has been recorded.

# Appendix B

## Survey Instrument 2

The following is the instrument that was created to assess the attributes in chapter 4, Study 2: Context-Aware Palette for a Block-Based Language.

QS\_IDE\_IRB

ExpertReview score Fair

Consent

Q1

**Block-Based IDE**  
Howard Hughes School of Engineering  
**Computer Science**  
**UNLV**

**INFORMED CONSENT FORM:**

Before you consider the research, you should be aware of the following information:

- Research is voluntary. You do not have to be in this research study.
- There are no risks to you from participating in this research study except for your time and inconvenience.
- You will fill out a survey with basic demographic information about yourself.
- You will look at an Integrated Development Environment (IDE) and answer questions about it.
- The research study will take approximately 15 minutes.
- If you agree to be in the study, you should read the rest of this document. The document explains what will happen to people in the study.
- You must be at least 18 years of age to participate.

**WHY ARE YOU BEING ASKED TO PARTICIPATE:**  
The purpose of the research is to learn about how block-based programming languages are used by practitioners.

**WHAT YOU WILL BE ASKED TO DO:**  
If you decide to participate, you will be asked to fill out a demographic survey (e.g., age, gender, etc), work through some problems related to reading software programs, and provide feedback.

**RISKS:**  
Except for your time and inconvenience, there are no risks to you from participating in this study.

**BENEFITS:**  
While this study will have no direct benefit to you, this research may help you learn more about programming and programming environments.

**CONFIDENTIALITY:**  
Your name will not be on any of the data. Your name or other identifying information will not be reported in any publications. The de-identified data could be used for future research studies or distributed to another investigator for future research studies without additional informed consent from you.

**VOLUNTARY:**  
Participation is voluntary. If you choose to take part in this study, you may stop at any time.

**CONTACT INFORMATION:**  
If you have any questions about this study, please contact Alex Hoffman at alex.hoffman@unlv.edu or Dr. Andreas Stefik at andreas.stefik@unlv.edu. If you have any questions about your rights as a research participant, please contact the Office of Research Integrity, University of Nevada Las Vegas, at ORI@unlv.edu.

If you click Accept, it indicates that you have read the above information and agree to participate, and you will continue to the survey.

Q2

Do you consent to be in this study?

Yes, I Accept

No, I Do Not Accept

Import from library Add new question

Add Block

Time

**Q1**

This survey should take approximately 15 minutes to complete.

Your responses are anonymous, so your answers cannot be associated to you.

First, you will be asked some demographics questions.

After that, you will be asked questions regarding the study. Answer each question to the best of your ability without using external references.

[Import from library](#) [Add new question](#)

[Add Block](#)

**Demographics**

**Q1** \*  
What is the highest level of education you **completed**?

- High School
- Some College
- Associate Degree
- Bachelors Degree
- Masters Degree
- PhD
- Other

**Q2** \*  
If you are currently enrolled in college, what is your classification?

- Freshman
- Sophomore
- Junior
- Senior
- Graduate Program
- Not Enrolled

**Q3** \*  
How old are you?

**Q4** \*  
What is your gender?

- Male
- Female
- Non-binary / third gender
- Prefer not to say

**Q5** \*  
What is your ethnicity?

- White
- Black or African American
- American Indian or Alaska Native
- Asian
- Native Hawaiian or Pacific Islander
- Other
- Prefer not to say

**Q6** \*

Are you now or have you ever been employed to write code professionally, regardless of the programming language?

- Currently employed to write code
- Previously employed to write code
- Never employed to write code as a primary job function, but it was a periodic part of the job
- Never employed to write code

**Q7** \*

What programming languages have you used to write code?

- C / C++
- Java
- Javascript
- HTML / CSS
- PHP
- Python
- Other
- None

**Q8** 👁️ \*

How many years of professional programming experience do you have?

**Q17**

Have you ever programmed in a **Block Based Language** such as Scratch, Blockly, Snap!, Lego Mindstorms, etc?

- Yes
- No

**Q17a**

How many years of experience do you have programming in a **Block Based Language** such as Scratch, Blockly, Snap!, Lego Mindstorms, etc?

- 0-3
- 3-6
- 6-9
- 9+

**Q18**

Have you ever programmed using an **Integrated Development Environment (IDE)** such as VS Code, IntelliJ, Eclipse, pyCharm, etc?

- Yes
- No

**Q18a**

How many years of experience do you have programming with an **Integrated Development Environment (IDE)** such as VS Code, IntelliJ, Eclipse, pyCharm, etc?

- 0-3
- 3-6
- 6-9
- 9+

[Import from library](#) [Add new question](#)

[Add Block](#)



Explanation

Q1

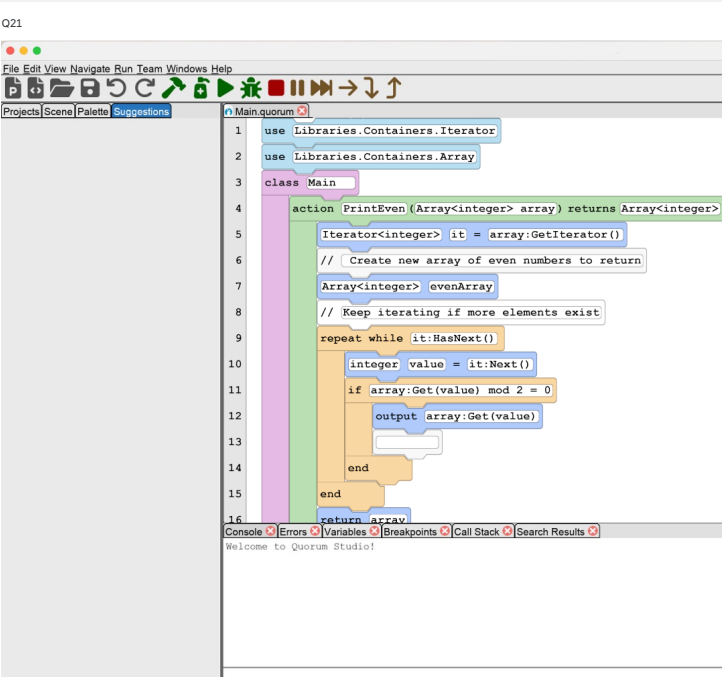
You will be introduced to an Integrated Development Environment (IDE) for a block-based programming language. You will then be asked some questions about the design of the IDE.

Import from library Add new question

Add Block

Setup

Q21



```

1 use Libraries.Containers.Iterator
2 use Libraries.Containers.Array
3 class Main
4   action PrintEven (Array<integer> array) returns Array<integer>
5     Iterator<integer> it = array.GetIterator()
6     // Create new array of even numbers to return
7     Array<integer> evenArray
8     // Keep iterating if more elements exist
9     repeat while it.HasNext()
10      integer value = it.Next()
11      if array.Get(value) mod 2 = 0
12        output array.Get(value)
13      end
14    end
15  end
16  return evenArray

```

Console Errors Variables Breakpoints Call Stack Search Results

Welcome to Quorum Studio!

In this image, you can see a partial implementation for a program. Right now, it does not matter what the purpose of this program is.

The main pane (code area) on the top right contains the blocks for the program code. Here you can see various types of blocks such as:

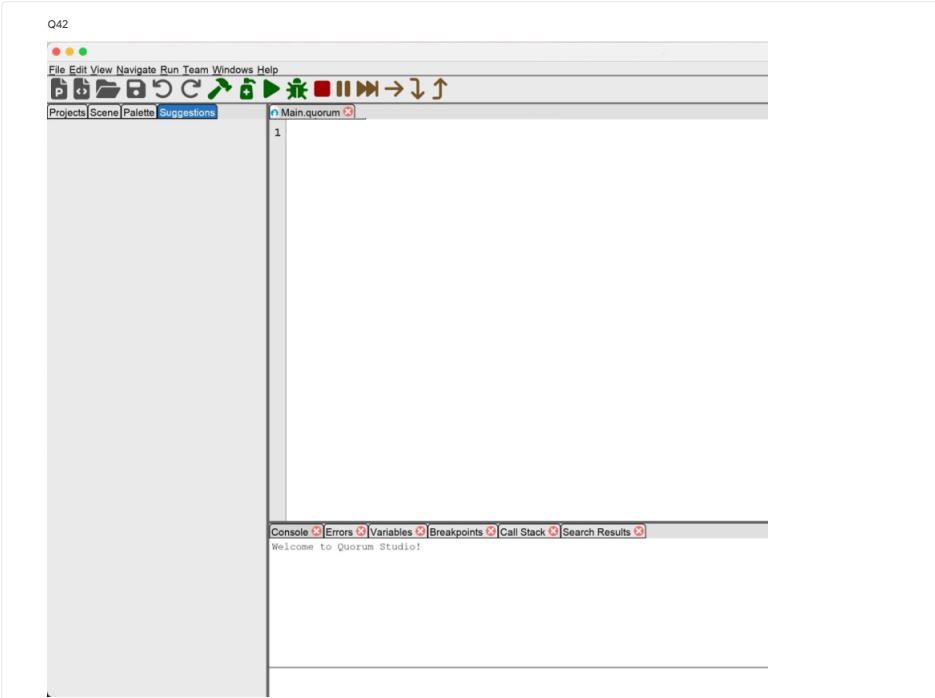
- Blocks to import: "use" to import a library
- Blocks to create new blocks: "class" and "action"
- Control: "repeat" blocks for repetition / looping
- Conditional: "if" block
- I/O: input/output blocks to print to the console or receive input from the user
- Exit: "return" block to return the array variable back to a calling function
- Variable blocks in blue
- Comment blocks in white
- Blank block to represent an empty line for spacing purposes

The bottom pane (which above, says "Welcome to Quorum Studio!") has the console with output when the program runs, and it can also show compiler errors, etc.

Import from library Add new question

Add Block

freeform\_setup



In this image, you can see a freshly opened IDE with an empty code editor.

The **left pane** contains section tabs such as Projects (current open projects and files), Scene, Palette, and Suggestions.

In the **left pane**, you can see the **Suggestions** tab. The **Suggestions** tab will provide elements that will help you write your program.

Q22

Imagine you need to write a new program. It could be anything like a recursive fibonacci function, a new game, or some statistical analysis.

In a broad sense, what could show up in the **Suggestions tab in the left pane** that would help you with writing a program?

[Import from library](#) [Add new question](#)

[Add Block](#)

q1\_use

Q39

```
1 use Libraries.Containers.Array ←
2 use Libraries.Containers.Iterator
```

Imagine you've been tasked to write a small function that will take in an integer array, prints out the even integers, and then returns an array of the even integers to the caller.

Assume that the cursor is anywhere on the line with the green arrow.

Think about which *items* below could be shown in the **left pane** that would be beneficial to your success in writing this program.

Q38

Please drag-and-drop the *items* below to **categorize** them into "HELPFUL in this context" or "NOT HELPFUL in this context."

Then, please **rank the items in the "HELPFUL in this context" category**, starting with 1=most helpful. You do not need to rank the items in the "NOT HELPFUL" category:

Items	HELPFUL in this context	NOT HELPFUL in this context
Project Files / Structure		
Alert About What Problems (red blocks) Mean & How to Resolve		
Source Control		
Class Name/Library		
Class Short Description (max 240 Chars)		
Link to Class Documentation		
Example Code		
Blocks of In-Scope Variables		
Blocks to Create New Variables		
Blocks for Control (Repetition / Loops)		
Blocks for Conditionals (if / else, etc)		
Blocks of All Functions in the Class		
Blocks for I/O (Print, Input, Clicks, etc)		
Blocks to Create New Blocks (New Class, New Function, etc)		
Blocks for Exiting (Return, Continue, Break, etc)		
Blocks to Import Additional Libraries (use)		

Q54

If you have any other helpful ideas, please enter them in the box below:

[Import from library](#) [Add new question](#)

[Add Block](#)

q2\_error

```
Q48
1 use Libraries.Containers.Iterator
2 class Main
3   action PrintEven (Array<integer> array) ←
4
5   end
6 end
```

Imagine you've been tasked to write a small function that will take in an integer array, prints out the even integers, and then returns an array of the even integers to the caller.

When a block is red, it means there is an error on the line.

Assume that the cursor is anywhere on the line with the green arrow.

Think about which items below could be shown in the left pane that would be beneficial to your success in writing this program.

Q49

Please drag-and-drop the *items* below to categorize them into "HELPFUL in this context" or "NOT HELPFUL in this context."

Then, please rank the items in the "HELPFUL in this context" category, starting with 1=most helpful. You do not need to rank the items in the "NOT HELPFUL" category.

Items	HELPFUL in this context	NOT HELPFUL in this context
Project Files / Structure		
Alert About What Problems (red blocks) Mean & How to Resolve		
Source Control		
Class Name/Library		
Class Short Description (max 240 Chars)		
Link to Class Documentation		
Example Code		
Blocks of In-Scope Variables		
Blocks to Create New Variables		
Blocks for Control (Repetition / Loops)		
Blocks for Conditionals (if / else, etc)		
Blocks of All Functions in the Class		
Blocks for I/O (Print, input, Clicks, etc)		
Blocks to Create New Blocks (New Class, New Function, etc)		
Blocks for Exiting (Return, Continue, Break, etc)		
Blocks to Import Additional Libraries (use)		

Q55

If you have any other helpful ideas, please enter them in the box below:

Import from library Add new question

Add Block

q3\_if\_block

```
Q42
1 use Libraries.Containers.Array
2 use Libraries.Containers.Iterator
3 class Main
4   action PrintEven (Array<integer> array)
5     Iterator<integer> it = array.GetIterator()
6     repeat while it.HasNext()
7       integer value = it.Next()
8       if
9     end
10  end
11 end
12
13
14 action Main
15
16 end
17 end
```

Imagine you've been tasked to write a small function that will take in an integer array, prints out the even integers, and then returns an array of the even integers to the caller.

Assume that the cursor is in the box with the "if" statement on the line with the green arrow.

Think about which items below could be shown in the left pane that would be beneficial to your success in writing this program.

Q43



Please drag-and-drop the items below to categorize them into "HELPFUL in this context" or "NOT HELPFUL in this context."

Then, please rank the items in the "HELPFUL in this context" category, starting with 1=most helpful. You do not need to rank the items in the "NOT HELPFUL" category:

Items	HELPFUL in this context	NOT HELPFUL in this context
Project Files / Structure		
Alert About What Problems (red blocks) Mean & How to Resolve		
Source Control		
Class Name/Library		
Class Short Description (max 240 Chars)		
Link to Class Documentation		
Example Code		
Blocks of In-Scope Variables		
Blocks to Create New Variables		
Blocks for Control (Repetition / Loops)		
Blocks for Conditionals (if / else, etc)		
Blocks of All Functions in the Class		
Blocks for I/O (Print, Input, Clicks, etc)		
Blocks to Create New Blocks (New Class, New Function, etc)		
Blocks for Exiting (Return, Continue, Break, etc)		
Blocks to import Additional Libraries (use)		

Q57

If you have any other helpful ideas, please enter them in the box below:

Import from library Add new question

Add Block

q4\_functionCall

Q50

```
1 action Main
2   Add(
3 end
4 action Add(integer a, integer b)
5   output a + b
6 end
```

Imagine you've been tasked to write a small function that takes in two integers and adds them together.

Assume that the cursor is at the end of the line with the green arrow, and you need to complete the function call.

Think about which *items* below could be shown in the **left pane** that would be beneficial to your success in writing this program.

Q51 🔍 ☆

Please drag-and-drop the *items* below to **categorize** them into "HELPFUL in this context" or "NOT HELPFUL in this context."

Then, please **rank the items in the "HELPFUL in this context" category**, starting with 1=most helpful. You do not need to rank the items in the "NOT HELPFUL" category:

Items	HELPFUL in this context	NOT HELPFUL in this context
Project Files / Structure		
Alert About What Problems (red blocks) Mean & How to Resolve		
Source Control		
Class Name/Library		
Class Short Description (max 240 Chars)		
Link to Class Documentation		
Example Code		
Blocks of In-Scope Variables		
Blocks to Create New Variables		
Blocks for Control (Repetition / Loops)		
Blocks for Conditionals (if / else, etc)		
Blocks of All Functions in the Class		
Blocks for I/O (Print, Input, Clicks, etc)		
Blocks to Create New Blocks (New Class, New Function, etc)		
Blocks for Exiting (Return, Continue, Break, etc)		
Blocks to Import Additional Libraries (use)		

Q58 🔍

If you have any other helpful ideas, please enter them in the box below:

[Import from library](#) [Add new question](#)

[Add Block](#)

q5\_dataframe

Q54

```
1 use Libraries.Compute.Statistics.DataFrame
2 use Libraries.Interface.Controls.Charts.Histogram
3 // Create a DataFrame to hold the data.
4 DataFrame frame ←
5 // Load your data file into the frame.
```

Imagine you're doing some statistical analysis, and need to load data into a dataframe.

Assume that the cursor is on the line with the green arrow where a new dataframe is being instantiated.

Think about which *items* below could be shown in the **left pane** that would be beneficial to your success in writing this program.

Q55 🔍 ☆

Please drag-and-drop the *items* below to **categorize** them into "HELPFUL in this context" or "NOT HELPFUL in this context."

Then, please **rank the items in the "HELPFUL in this context" category**, starting with 1=most helpful. You do not need to rank the items in the "NOT HELPFUL" category:

Items	HELPFUL in this context	NOT HELPFUL in this context
Project Files / Structure		
Alert About What Problems (red blocks) Mean & How to Resolve		
Source Control		
Class Name/Library		
Class Short Description (max 240 Chars)		
Link to Class Documentation		
Example Code		
Blocks of In-Scope Variables		
Blocks to Create New Variables		
Blocks for Control (Repetition / Loops)		
Blocks for Conditionals (if / else, etc)		
Blocks of All Functions in the Class		
Blocks for I/O (Print, Input, Clicks, etc)		
Blocks to Create New Blocks (New Class, New Function, etc)		
Blocks for Exiting (Return, Continue, Break, etc)		
Blocks to Import Additional Libraries (use)		

Q59 🔍

If you have any other helpful ideas, please enter them in the box below:

[Import from library](#) [Add new question](#)

[Add Block](#)

▼ q6\_chart



O52

```
1 use Libraries.Compute.Statistics.DataFrame
2 use Libraries.Interface.Controls.Charts.Histogram
3 // Create a DataFrame to hold the data.
4 DataFrame frame
5 // Load your data file into the frame.
6 frame:Load("data/AB_NYC_2019.csv")
7 // Select data from the numerical column "price" for the histogram.
8 frame:AddSelectedColumns("price")
9 // Using the frame, create a Histogram object.
10 Histogram chart = frame:Histogram()
12 chart: ←
```

Imagine you're doing some statistical analysis, and you need to display the chart or make some edits to the chart.

Assume that the cursor is on the line with the green arrow after the colon ( : ) (similar to dot notation in Java or Python) and you need to complete this line of code.

Think about which *items* below could be shown in the **left pane** that would be beneficial to your success in writing this program.

O53



Please drag-and-drop the *items* below to **categorize** them into "HELPFUL in this context" or "NOT HELPFUL in this context."

Then, please **rank the items in the "HELPFUL in this context" category**, starting with 1=most helpful. You do not need to rank the items in the "NOT HELPFUL" category:

Items	HELPFUL in this context	NOT HELPFUL in this context
Project Files / Structure		
Alert About What Problems (red blocks) Mean & How to Resolve		
Source Control		
Class Name/Library		
Class Short Description (max 240 Chars)		
Link to Class Documentation		
Example Code		
Blocks of In-Scope Variables		
Blocks to Create New Variables		
Blocks for Control (Repetition / Loops)		
Blocks for Conditionals (if / else, etc)		
Blocks of All Functions in the Class		
Blocks for I/O (Print, Input, Clicks, etc)		
Blocks to Create New Blocks (New Class, New Function, etc)		
Blocks for Exiting (Return, Continue, Break, etc)		
Blocks to Import Additional Libraries (use)		

Q60

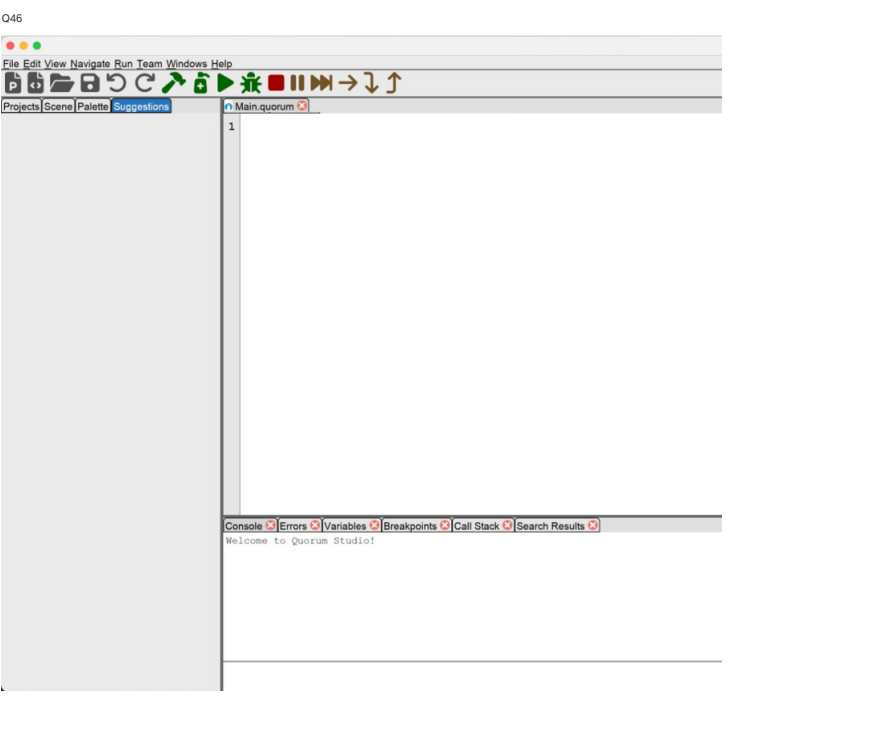
If you have any other helpful ideas, please enter them in the box below:

Import from library Add new question

Add Block

Re-ask-freeform-text

Q46



Q47

Now that you've had more time to think about it in context, imagine you need to write a new program. It could be anything like a recursive fibonacci function, a new game, or some statistical analysis.

In a broad sense, what could show up in the Suggestions tab that would help you with writing this function?

Import from library Add new question

Add Block

Final-ask-student

Q1 🗑️

Thank you for your time. If you have any additional feedback, please record it in the textbox below.

Q2 \*

If you wish to enter your email address to verify you took this survey for course credit, please select "Yes" to be directed to a new form. Your email address will not be associated with your survey responses.

Otherwise select "No" to end the survey without recording your email.

Yes

No

[Import from library](#) [Add new question](#)

[Add Block](#)

Professional Future Contact

Q1

Would you like to be contacted in the future with the results of this study or for opportunities to participate in future studies?

Yes

No

[Import from library](#) [Add new question](#)

[Add Block](#)

End of Survey

We thank you for your time spent taking this survey.

Your response has been recorded.

# Bibliography

- [1] E. P. Glinert, *Towards' second generation'interactive, graphical programming environments*. Department of Computer Science, Rensselaer Polytechnic Institute, 1986.
- [2] D. J. Malan and H. H. Leitner, "Scratch for budding computer scientists," in *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, ser. SIGCSE '07. Association for Computing Machinery, 2007, pp. 223–227. [Online]. Available: <https://dl.acm.org/doi/10.1145/1227310.1227388>
- [3] Google, "Blockly," <https://developers.google.com/blockly>, Accessed on November 24, 2023. [Online]. Available: <https://developers.google.com/blockly>
- [4] A. Zilberman and L. Ice, "Why computer occupations are behind strong stem employment growth in the 2019–29 decade," 2021, <https://www.bls.gov/opub/btn/volume-10/why-computer-occupations-are-behind-strong-stem-employment-growth.htm>, Accessed on September 21, 2023. [Online]. Available: <https://www.bls.gov/opub/btn/volume-10/why-computer-occupations-are-behind-strong-stem-employment-growth.htm>
- [5] B. A., O. A., and K. Hale, "The state of u.s. science and engineering 2022," 2022, <https://nces.nsf.gov/pubs/nsb20221/u-s-and-global-stem-education-and-labor-force>, Accessed on September 21, 2023. [Online]. Available: <https://nces.nsf.gov/pubs/nsb20221/u-s-and-global-stem-education-and-labor-force>
- [6] H. Berghel, "STEM, Revisited," *Computer*, vol. 47, no. 3, pp. 70–73, Mar. 2014. [Online]. Available: <https://ieeexplore.ieee.org/document/6766172>
- [7] H. Berghel, "STEM Crazy," *Computer*, vol. 48, no. 9, pp. 75–80, Sep. 2015. [Online]. Available: <https://ieeexplore.ieee.org/document/7274416>
- [8] D. Weintrop, "iSchools as Venues for Expanding the K-12 Computer Science Teacher Pipeline," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*. Providence RI USA: ACM, Feb. 2022, pp. 397–403. [Online]. Available: <https://dl.acm.org/doi/10.1145/3478431.3499302>
- [9] D. Weintrop and U. Wilensky, "Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms," *Computers & Education*, vol. 142, p. 103646, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S036013151930199X>
- [10] S. Papert, *Mindstorms: Children, Computers, and Powerful Ideas*. USA: Basic Books, Inc., 1980.
- [11] S. Papert, "Constructionism: A new opportunity for elementary science education," 1986, proposal to the National Science Foundation.

- [12] F. Martin and M. Resnick, “LEGO/Logo and Electronic Bricks: Creating a Scienceland for Children,” in *Advanced Educational Technologies for Mathematics and Science*, ser. NATO ASI Series, D. L. Ferguson, Ed. Berlin, Heidelberg: Springer, 1993, pp. 61–89.
- [13] J. M. Wing, “Computational thinking,” *Commun. ACM*, vol. 49, no. 3, p. 33–35, mar 2006. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/1118178.1118215>
- [14] M. A. Kuhail, S. Farooq, R. Hammad, and M. Bahja, “Characterizing visual programming approaches for end-user developers: A systematic review,” *IEEE Access*, vol. 9, pp. 14 181–14 202, 2021.
- [15] M. Tempel, “Blocks programming,” 2013, [https://el.media.mit.edu/logo-foundation/resources/papers/pdf/blocks\\_programming.pdf](https://el.media.mit.edu/logo-foundation/resources/papers/pdf/blocks_programming.pdf), Accessed on September 21, 2023. [Online]. Available: [https://el.media.mit.edu/logo-foundation/resources/papers/pdf/blocks\\_programming.pdf](https://el.media.mit.edu/logo-foundation/resources/papers/pdf/blocks_programming.pdf)
- [16] D. Weintrop, “Modality matters: Understanding the Effects of Programming Language Representation in High School Computer Science Classrooms,” Ph.D., Northwestern University, United States – Illinois, 2016, iSBN: 9781369154825. [Online]. Available: <https://www.proquest.com/pqlacademic/docview/1826352865/abstract/CA3518E64343B8PQ/1>
- [17] U. Wolz, H. H. Leitner, D. J. Malan, and J. Maloney, “Starting with scratch in cs 1,” in *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 2–3. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/1508865.1508869>
- [18] J. A. Martínez-Valdés, J. A. Velázquez-Iturbide, and R. Hijón-Neira, “A (relatively) unsatisfactory experience of use of scratch in cs1,” in *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality*, ser. TEEM 2017. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/3144826.3145356>
- [19] D. Weintrop and U. Wilensky, “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*, ser. IDC ’15. New York, NY, USA: Association for Computing Machinery, Jun. 2015, pp. 199–208. [Online]. Available: <https://doi.org/10.1145/2771839.2771860>
- [20] W. Dann, D. Cosgrove, D. Slater, D. Culyba, and S. Cooper, “Mediated transfer: Alice 3 to java,” in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’12. New York, NY, USA: Association for Computing Machinery, 2012, p. 141–146. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/2157136.2157180>
- [21] R. L. Bangert-Drowns, M. M. Hurley, and B. Wilkinson, “The Effects of School-Based Writing-to-Learn Interventions on Academic Achievement: A Meta-Analysis,” *Review of Educational Research*, vol. 74, no. 1, pp. 29–58, Mar. 2004, publisher: American Educational Research Association. [Online]. Available: <https://doi.org/10.3102/00346543074001029>
- [22] R. Ladner, “Expanding the pipeline: The status of persons with disabilities in the computer science pipeline,” Jan 2021, accessed on November 18, 2023. [Online]. Available: <https://cra.org/crn/2020/11/expanding-the-pipeline-the-status-of-persons-with-disabilities-in-the-computer-science-pipeline/>

- [23] N. Turner Lee, P. Resnick, and G. Barton, “Algorithmic bias detection and mitigation: Best practices and policies to reduce consumer harms,” May 2019, accessed on November 18, 2023. [Online]. Available: <https://www.brookings.edu/articles/algorithmic-bias-detection-and-mitigation-best-practices-and-policies-to-reduce-consumer-harms/>
- [24] C. Boutin, “There’s more to ai bias than biased data, nist report highlights,” Mar 2022, <https://www.nist.gov/news-events/news/2022/03/theres-more-ai-bias-biased-data-nist-report-highlights>, Accessed on November 18, 2023. [Online]. Available: <https://www.nist.gov/news-events/news/2022/03/theres-more-ai-bias-biased-data-nist-report-highlights>
- [25] P. G. Feijóo-García, S. Wang, J. Cai, N. Polavarapu, C. Gardner-McCune, and E. D. Ragan, “Design and evaluation of a scaffolded block-based learning environment for hierarchical data structures,” in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2019, pp. 145–149.
- [26] P. G. Feijóo-García, A. Kapoor, C. Gardner-McCune, and E. Ragan, “Effects of a block-based scaffolded tool on students’ introduction to hierarchical data structures,” *IEEE Transactions on Education*, vol. 65, no. 2, pp. 191–199, 2022.
- [27] K. Brennan and M. Resnick, “New frameworks for studying and assessing the development of computational thinking,” in *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, vol. 1, 2012, p. 25.
- [28] S. Grover, R. Pea, and S. Cooper, “Designing for deeper learning in a blended computer science course for middle school students,” *Computer Science Education*, vol. 25, no. 2, pp. 199–237, 2015, publisher: Routledge \_eprint: <https://doi.org/10.1080/08993408.2015.1033142>. [Online]. Available: <https://doi.org/10.1080/08993408.2015.1033142>
- [29] K. N. Whitley, “Visual programming languages and the empirical evidence for and against,” *Journal of Visual Languages & Computing*, vol. 8, no. 1, pp. 109–142, 1997. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1045926X96900300>
- [30] Epic, “Unreal blueprints,” <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints>, Accessed on October 07, 2023. [Online]. Available: <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints>
- [31] T. Sharma, S. Georgiou, M. Kechagia, T. A. Ghaleb, and F. Sarro, “Investigating developers’ perception on software testability and its effects,” *Empirical Software Engineering*, vol. 28, no. 5, p. 120, 2023. [Online]. Available: <https://doi.org/10.1007/s10664-023-10373-0>
- [32] K. N. Whitley and A. F. Blackwell, “Visual programming: the outlook from academia and industry,” in *Papers presented at the seventh workshop on Empirical studies of programmers*, ser. ESP ’97. Association for Computing Machinery, 1997, pp. 180–208. [Online]. Available: <https://dl.acm.org/doi/10.1145/266399.266415>
- [33] A. Bragdon, R. Zeleznik, S. P. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, “Code bubbles: a working set-based interface for code understanding and maintenance,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’10. Association for Computing Machinery, 2010, pp. 2503–2512. [Online]. Available: <https://dl.acm.org/doi/10.1145/1753326.1753706>

- [34] L. Gewali, personal communication, Apr. 2024.
- [35] S. Tanimoto and E. Glinert, "Pict: An interactive graphical programming environment," *Computer*, vol. 17, no. 11, pp. 7–25, nov 1984.
- [36] J. Vento, "Application of labview in higher education laboratories," in *Proceedings Frontiers in Education Conference*, 1988, pp. 444–447.
- [37] D. M. Gee, "Formal specification of visual languages," *Information and Software Technology*, vol. 40, no. 7, pp. 359–367, 1998. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584998000597>
- [38] J. V. Nickerson, *Visual programming*. New York University, 1994, <http://www.nickerson.to/visprog/visprog.htm>, Accessed on September 2, 2023.
- [39] M. A. Jackson, "Constructive methods of program design," in *ECI Conference 1976*, ser. Lecture notes in computer science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1976, pp. 236–262.
- [40] W. R. Sutherland, "The on-line graphical specification of computer procedures." Thesis, Massachusetts Institute of Technology, 1966, accepted: 2005-09-21T22:40:43Z. [Online]. Available: <https://dspace.mit.edu/handle/1721.1/13474>
- [41] MIT, "Logo history," [https://el.media.mit.edu/logo-foundation/what\\_is\\_logo/history.html](https://el.media.mit.edu/logo-foundation/what_is_logo/history.html), Accessed on November 05, 2023. [Online]. Available: [https://el.media.mit.edu/logo-foundation/what\\_is\\_logo/history.html](https://el.media.mit.edu/logo-foundation/what_is_logo/history.html)
- [42] P. Boytchev, "The logo tree project," <https://pavel.it.fmi.uni-sofia.bg/logotree/table.html>, Accessed on November 05, 2023. [Online]. Available: <https://pavel.it.fmi.uni-sofia.bg/logotree/table.html>
- [43] A. Begel, "Logoblocks: A graphical programming language for interacting with the world," *Electrical Engineering and Computer Science Department, MIT, Boston, MA*, vol. 2, 1996.
- [44] B. A. Myers, "Visual programming, programming by example, and program visualization: A taxonomy," *SIGCHI Bull.*, vol. 17, no. 4, p. 59–66, apr 1986. [Online]. Available: <https://doi.org/10.1145/22339.22349>
- [45] J. H. Kaas and P. Balaram, "Current research on the organization and function of the visual system in primates." 2014.
- [46] L. R. Milne, "Blocks4All: making block programming languages accessible for blind children," *ACM SIGACCESS Accessibility and Computing*, no. 117, pp. 26–29, Feb. 2017. [Online]. Available: <https://doi.org/10.1145/3051519.3051525>
- [47] A. Stefik, W. Allee, G. Contreras, T. Kluthe, A. Hoffman, B. Blaser, and R. Ladner, "Accessible to whom? bringing accessibility to blocks," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 1286–1292. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/3626252.3630770>
- [48] A. Mountapmbeme, O. Okafor, and S. Ludi, "Accessible Blockly: An Accessible Block-Based Programming Library for People with Visual Impairments," in *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS '22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 1–15. [Online]. Available: <https://doi.org/10.1145/3517428.3544806>

- [49] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, “Scratch: programming for all,” *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009. [Online]. Available: <https://dl.acm.org/doi/10.1145/1592761.1592779>
- [50] S. Cooper, W. Dann, and R. Pausch, “Alice: A 3-d tool for introductory programming concepts,” *Journal of Computing Sciences in Colleges - JCSC*, vol. 15, no. 5, p. 107–116, 01 2000.
- [51] E. Coronado, F. Mastrogiovanni, B. Indurkha, and G. Venture, “Visual programming environments for end-user development of intelligent and social robots, a systematic review,” *Journal of Computer Languages*, vol. 58, p. 100970, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2590118420300307>
- [52] Unity, “Unity visual scripting,” 2023, <https://unity.com/features/unity-visual-scripting>, Accessed on October 07, 2023. [Online]. Available: <https://unity.com/features/unity-visual-scripting>
- [53] M. Mohd Azmi, K. Lim, P. Mohan, and T. Kit, “UNITEN Smart Programming Apps Using Combination of Robotic and Block Programming (RoBlock),” in *2021 IEEE International Conference on Computing (ICOCO)*, Nov. 2021, pp. 202–207.
- [54] A. C. Bart, J. Tibau, D. Kafura, C. A. Shaffer, and E. Tilevich, “Design and Evaluation of a Block-based Environment with a Data Science Context,” *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 1, pp. 182–192, Jan. 2020, conference Name: IEEE Transactions on Emerging Topics in Computing.
- [55] L. Barboza, R. Mello, M. Modell, and E. S. Teixeira, “Blockly-DS: Blocks Programming for Data Science with Visual, Statistical, Descriptive and Predictive Analysis,” in *LAK23: 13th International Learning Analytics and Knowledge Conference*, ser. LAK2023. New York, NY, USA: Association for Computing Machinery, Mar. 2023, pp. 644–649. [Online]. Available: <https://doi.org/10.1145/3576050.3576097>
- [56] P. d. L. Sobreira, J. W. Abijaude, H. D. G. Viana, L. M. S. Santiago, K. E. Guemhioui, O. A. Wahab, and F. Greve, “Usability evaluation of block programming tools in IoT contexts for initial engineering courses,” in *2020 IEEE World Conference on Engineering Education (EDUNINE)*, Mar. 2020, pp. 1–5.
- [57] W. Underwood, D. Weintrop, M. Kurtz, and R. Marciano, “Introducing Computational Thinking into Archival Science Education,” in *2018 IEEE International Conference on Big Data (Big Data)*. Seattle, WA, USA: IEEE, Dec. 2018, pp. 2761–2765. [Online]. Available: <https://ieeexplore.ieee.org/document/8622511/>
- [58] A. Feng, E. Tilevich, and W.-c. Feng, “Block-based programming abstractions for explicit parallel computing,” in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, Oct. 2015, pp. 71–75.
- [59] A. L. V. Solórzano and A. S. Charão, “BlocklyPar: from sequential to parallel with block-based visual programming,” in *2021 IEEE Frontiers in Education Conference (FIE)*, Oct. 2021, pp. 1–8, iSSN: 2377-634X.
- [60] M. Verano Merino, J. P. Sáenz, and A. M. Díaz Castillo, “Suppose You Had Blocks within a Notebook,” in *Proceedings of the 1st ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments*, ser. PAINT 2022. New York,



- NY, USA: Association for Computing Machinery, Dec. 2022, pp. 57–62. [Online]. Available: <https://doi.org/10.1145/3563836.3568728>
- [61] M. Verano Merino and K. van Wijk, “Workbench for Creating Block-Based Environments,” in *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering*, ser. SLE 2022. New York, NY, USA: Association for Computing Machinery, Dec. 2022, pp. 61–73. [Online]. Available: <https://doi.org/10.1145/3567512.3567518>
- [62] M. Kazemitabaar, V. Chyhir, D. Weintrop, and T. Grossman, “Scaffolding Progress: How Structured Editors Shape Novice Errors When Transitioning from Blocks to Text,” *SIGCSE 2023: Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, 2023.
- [63] “Web content accessibility guidelines (wcag) 2.2,” <https://www.w3.org/TR/WCAG22/>, Accessed on Oct 04, 2023. [Online]. Available: <https://www.w3.org/TR/WCAG22/>
- [64] T. R. Green and M. Petre, “When visual programs are harder to read than textual programs,” in *Human-Computer Interaction: Tasks and Organisation, Proceedings ECCE-6 (6th European Conference Cognitive Ergonomics)*, vol. 57, 1992.
- [65] H. Berghel, “New wave prototyping: use and abuse of vacuous prototypes,” *interactions*, vol. 1, no. 2, pp. 49–54, 1994.
- [66] B. Harvey, “Bringing “no ceiling” to scratch: Can one language serve kids and computer scientists?” in *Bringing “No Ceiling” to Scratch: Can One Language Serve Kids and Computer Scientists?*, 2010, <https://api.semanticscholar.org/CorpusID:62609287>. [Online]. Available: <https://api.semanticscholar.org/CorpusID:62609287>
- [67] N. Fraser, “Ten things we’ve learned from blockly,” in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 2015, pp. 49–50.
- [68] Code.org, “Learn computer science. change the world.” <https://code.org/>, Accessed on October 02, 2023. [Online]. Available: <https://code.org/>
- [69] J. Devine, J. Finney, P. de Halleux, M. Moskal, T. Ball, and S. Hodges, “Makecode and codal: Intuitive and efficient embedded systems programming for education,” in *Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, ser. LCTES 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 19–30. [Online]. Available: <https://doi.org/10.1145/3211332.3211335>
- [70] C. Scaffidi and C. Chambers, “Skill progression demonstrated by users in the scratch animation environment,” *International Journal of Human-Computer Interaction*, vol. 28, no. 6, pp. 383–398, 2012. [Online]. Available: <https://doi.org/10.1080/10447318.2011.595621>
- [71] P. Grabarczyk, S. M. Nicolajsen, and C. Brabrand, “On the effect of onboarding computing students without programming-confidence or -experience,” in *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research*, ser. Koli Calling ’22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/3564721.3564724>
- [72] D. Hagan and S. Markham, “Does it help to have some programming experience before beginning a computing degree program?” in *Proceedings of the 5th Annual SIGCSE/SIGCUE*

- ITiCSEconference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 25–28. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/343048.343063>
- [73] J. R. Davy, K. Audin, M. Barkham, and C. Joyner, “Student well-being in a computing department,” in *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSEconference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 136–139. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/343048.343145>
- [74] C. Alvarado, G. Umbelino, and M. Minnes, “The persistent effect of pre-college computing experience on college cs course grades,” in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 876–881. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/3159450.3159508>
- [75] G. Bui, N. Sibia, A. Zavaleta Bernuy, M. Liut, and A. Petersen, “Prior programming experience: A persistent performance gap in cs1 and cs2,” in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 889–895. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/3545945.3569752>
- [76] S. Krause-Levy, S. Valstar, L. Porter, and W. G. Griswold, “A demographic analysis on prerequisite preparation in an advanced data structures course,” in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1*, ser. SIGCSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 661–667. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/3478431.3499337>
- [77] O. Meerbaum-Salant, M. Armoni, and M. M. Ben-Ari, “Learning computer science concepts with scratch,” in *Proceedings of the Sixth International Workshop on Computing Education Research*, ser. ICER '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 69–76. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/1839594.1839607>
- [78] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, “Habits of programming in scratch,” in *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 168–172. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/1999747.1999796>
- [79] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, “From scratch to "real" programming,” *ACM Trans. Comput. Educ.*, vol. 14, no. 4, Feb 2015. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/2677087>
- [80] C. M. Lewis, “How programming environment shapes perception, learning and goals: Logo vs. scratch,” in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 346–350. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/1734263.1734383>
- [81] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, “The scratch programming language and environment,” *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 16:1–16:15, 2010. [Online]. Available: <https://dl.acm.org/doi/10.1145/1868358.1868363>

- [82] S. Cooper, “The design of alice,” *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 15:1–15:16, 2010. [Online]. Available: <https://dl.acm.org/doi/10.1145/1868358.1868362>
- [83] T. W. Price and T. Barnes, “Comparing Textual and Block Interfaces in a Novice Programming Environment,” in *Proceedings of the eleventh annual International Conference on International Computing Education Research*, ser. ICER ’15. New York, NY, USA: Association for Computing Machinery, Aug. 2015, pp. 91–99. [Online]. Available: <https://doi.org/10.1145/2787622.2787712>
- [84] D. Weintrop and U. Wilensky, “Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms,” *ACM Transactions on Computing Education*, vol. 18, no. 1, pp. 3:1–3:25, Oct. 2017. [Online]. Available: <https://doi.org/10.1145/3089799>
- [85] M. Seraj, E.-S. Katterfeldt, K. Bub, S. Autexier, and R. Drechsler, “Scratch and Google Blockly: How Girls’ Programming Skills and Attitudes are Influenced,” in *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling ’19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 1–10. [Online]. Available: <https://doi.org/10.1145/3364510.3364515>
- [86] D. Sun, C. Zhu, F. Xu, Y. Li, F. Ouyang, and M. Chen, “Transitioning from introductory to professional programming in secondary education: Comparing learners’ computational thinking skills, behaviors, and attitudes,” *Journal of Educational Computing Research*, p. 07356331231204653, 2023, publisher: SAGE Publications Inc. [Online]. Available: <https://doi.org/10.1177/07356331231204653>
- [87] H. Dwyer, C. Hill, A. Hansen, A. Iveland, D. Franklin, and D. Harlow, “Fourth Grade Students Reading Block-Based Programs: Predictions, Visual Cues, and Affordances,” in *Proceedings of the eleventh annual International Conference on International Computing Education Research*, ser. ICER ’15. New York, NY, USA: Association for Computing Machinery, Aug. 2015, pp. 111–119. [Online]. Available: <https://doi.org/10.1145/2787622.2787729>
- [88] A. Stefik and E. Gellenbeck, “Empirical studies on programming language stimuli,” *Software Quality Journal*, vol. 19, no. 1, pp. 65–99, Mar. 2011. [Online]. Available: <https://doi.org/10.1007/s11219-010-9106-7>
- [89] A. Stefik and S. Siebert, “An empirical investigation into programming language syntax,” *ACM Trans. Comput. Educ.*, vol. 13, no. 4, nov 2013. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/2534973>
- [90] D. Weintrop, “Minding the Gap Between Blocks-Based and Text-Based Programming (Abstract Only),” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’15. New York, NY, USA: Association for Computing Machinery, Feb. 2015, p. 720. [Online]. Available: <https://doi.org/10.1145/2676723.2693622>
- [91] R. E. Mayer, “Should There Be a Three-Strikes Rule Against Pure Discovery Learning?” *American Psychologist*, vol. 59, no. 1, pp. 14–19, 2004. [Online]. Available: <http://doi.apa.org/getdoi.cfm?doi=10.1037/0003-066X.59.1.14>
- [92] P. Techapalokul and E. Tilevich, “Understanding recurring quality problems and their impact on code sharing in block-based software,” in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct. 2017, pp. 43–51, iSSN: 1943-6106.

- [93] P. Techapalokul, “Sniffing Through Millions of Blocks for Bad Smells,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE ’17. New York, NY, USA: Association for Computing Machinery, Mar. 2017, pp. 781–782. [Online]. Available: <https://doi.org/10.1145/3017680.3022450>
- [94] E. Aivaloglou and F. Hermans, “How Kids Code and How We Know: An Exploratory Study on the Scratch Repository,” in *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ser. ICER ’16. New York, NY, USA: Association for Computing Machinery, Aug. 2016, pp. 53–61. [Online]. Available: <https://doi.org/10.1145/2960310.2960325>
- [95] L. Moors, A. Luxton-Reilly, and P. Denny, “Transitioning from Block-Based to Text-Based Programming Languages,” in *2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, Apr. 2018, pp. 57–64, iSSN: 2475-1057.
- [96] A. Mountapmbeme and S. Ludi, “How Teachers of the Visually Impaired Compensate with the Absence of Accessible Block-Based Languages,” in *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS ’21. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 1–10. [Online]. Available: <https://doi.org/10.1145/3441852.3471221>
- [97] A. Mountapmbeme, O. Okafor, and S. Ludi, “Accessible Blockly: An Accessible Block-Based Programming Library for People with Visual Impairments,” in *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS ’22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 1–15. [Online]. Available: <https://doi.org/10.1145/3517428.3544806>
- [98] Z. Wang and A. Wagner, “Evaluating a Tactile Approach to Programming Scratch,” in *Proceedings of the 2019 ACM Southeast Conference*, ser. ACM SE ’19. New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 226–229. [Online]. Available: <https://dl.acm.org/doi/10.1145/3299815.3314464>
- [99] J. Asbell-Clarke, T. Robillard, T. Edwards, E. Bardar, D. Weintrop, S. Grover, and M. Israel, “Including Neurodiversity in Foundational and Applied Computational Thinking (INFACT),” in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*, ser. SIGCSE 2022. New York, NY, USA: Association for Computing Machinery, Mar. 2022, p. 1076. [Online]. Available: <https://dl.acm.org/doi/10.1145/3478432.3499044>
- [100] A. Ismail, N. Omar, and A. Mohd Zin, “Developing learning software for children with learning disabilities through Block-Based development approach,” in *2009 International Conference on Electrical Engineering and Informatics*, vol. 01, Aug. 2009, pp. 299–303, iSSN: 2155-6830.
- [101] O. Okafor and S. Ludi, “Voice-Enabled Blockly: Usability Impressions of a Speech-driven Block-based Programming System,” in *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS ’22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 1–5. [Online]. Available: <https://doi.org/10.1145/3517428.3550382>
- [102] D. Weintrop, “Block-based programming in computer science education,” *Commun. ACM*, vol. 62, no. 8, p. 22–25, jul 2019. [Online]. Available: <https://doi.org/10.1145/3341221>
- [103] H. Alrubaye, S. Ludi, and M. W. Mkaouer, “Comparison of block-based and hybrid-based environments in transferring programming skills to text-based environments,” in *Proceedings of the 29th*

*Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '19. USA: IBM Corp., Nov. 2019, pp. 100–109.

- [104] J. N. Matias, S. Dasgupta, and B. M. Hill, “Skill Progression in Scratch Revisited,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 1486–1490. [Online]. Available: <https://doi.org/10.1145/2858036.2858349>
- [105] D. M. Kurland and R. D. Pea, “Children’s mental models of recursive logo programs,” *Journal of Educational Computing Research*, vol. 1, no. 2, pp. 235–243, 1985, publisher: SAGE Publications Inc. [Online]. Available: <https://doi.org/10.2190/JV9Y-5PD0-MX22-9J4Y>
- [106] D. Parsons and P. Haden, “Programming osmosis: Knowledge transfer from imperative to visual programming environments,” in *Proceedings of The Twentieth Annual NACCQ Conference*, Hamilton New Zealand, 2007, vol. 209.
- [107] P. Kinnunen and L. Malmi, “Why students drop out CS1 course?” in *Proceedings of the second international workshop on Computing education research*, ser. ICER '06. New York, NY, USA: Association for Computing Machinery, Sep. 2006, pp. 97–108. [Online]. Available: <https://dl.acm.org/doi/10.1145/1151588.1151604>
- [108] D. Bau, “Droplet, a blocks-based editor for text code,” *J. Comput. Sci. Coll.*, vol. 30, no. 6, p. 138–144, jun 2015.
- [109] D. Bau, D. A. Bau, M. Dawson, and C. S. Pickens, “Pencil code: Block code for a text world,” in *Proceedings of the 14th International Conference on Interaction Design and Children*, ser. IDC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 445–448. [Online]. Available: <https://doi.org/10.1145/2771839.2771875>
- [110] D. Weintrop and U. Wilensky, “Between a Block and a Typeface: Designing and Evaluating Hybrid Programming Environments,” in *Proceedings of the 2017 Conference on Interaction Design and Children*, ser. IDC '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 183–192. [Online]. Available: <https://doi.org/10.1145/3078072.3079715>
- [111] M. Krafft, G. Fraser, and N. Walkinshaw, “Motivating Adult Learners by Introducing Programming Concepts with Scratch,” in *Proceedings of the 4th European Conference on Software Engineering Education*, ser. ECSEE '20. New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 22–26. [Online]. Available: <https://dl.acm.org/doi/10.1145/3396802.3396818>
- [112] G. Weber, “Code is not just text: Why our code editors are inadequate tools,” in *Companion Proceedings of the 1st International Conference on the Art, Science, and Engineering of Programming*, ser. Programming '17. Association for Computing Machinery, 2017, pp. 1–3. [Online]. Available: <https://dl.acm.org/doi/10.1145/3079368.3079415>
- [113] M. Lodi, “Informatical Thinking,” *OLYMPIADS IN INFORMATICS*, pp. 113–132, Dec. 2020. [Online]. Available: [https://ioinformatics.org/journal/v14\\_2020\\_113\\_132.pdf](https://ioinformatics.org/journal/v14_2020_113_132.pdf)
- [114] L. Zhang and J. Nouri, “A systematic review of learning computational thinking through scratch in k-9,” *Computers & Education*, vol. 141, p. 103607, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0360131519301605>

- [115] J. M. Wing and D. Stanzione, “Progress in computational thinking, and expanding the hpc community,” *Commun. ACM*, vol. 59, no. 7, p. 10–11, jun 2016. [Online]. Available: <https://doi.org/10.1145/2933410>
- [116] S. Grover and R. Pea, “Computational Thinking in K–12: A Review of the State of the Field,” *Educational Researcher*, vol. 42, no. 1, pp. 38–43, Jan. 2013. [Online]. Available: <http://journals.sagepub.com/doi/10.3102/0013189X12463051>
- [117] L. Blum and T. J. Cortina, “Cs4hs: An outreach program for high school cs teachers,” *SIGCSE Bull.*, vol. 39, no. 1, p. 19–23, mar 2007. [Online]. Available: <https://doi.org/10.1145/1227504.1227320>
- [118] A. Siraj, M. J. Kosa, and S.-M. Olmstead, “Weaving a tapestry: Creating a satellite workshop to support hs cs teachers in attracting and engaging students,” in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’12. New York, NY, USA: Association for Computing Machinery, 2012, p. 493–498. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/2157136.2157282>
- [119] H. Bort and D. Brylow, “Cs4impact: Measuring computational thinking concepts present in cs4hs participant lesson plans,” in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’13. New York, NY, USA: Association for Computing Machinery, 2013, p. 427–432. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/2445196.2445323>
- [120] J. M. Doderer, J. M. Mota, and I. Ruiz-Rube, “Bringing computational thinking to teachers’ training: a workshop review,” in *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality*, ser. TEEM 2017. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1–6. [Online]. Available: <https://dl.acm.org/doi/10.1145/3144826.3145352>
- [121] N. Tumlin, “Teacher Configurable Coding Challenges for Block Languages,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE ’17. New York, NY, USA: Association for Computing Machinery, Mar. 2017, pp. 783–784. [Online]. Available: <https://doi.org/10.1145/3017680.3022467>
- [122] S. Grover, V. Cateté, T. Barnes, M. Hill, A. Ledeczi, and B. Broll, “FIRST Principles to Design for Online, Synchronous High School CS Teacher Training and Curriculum Co-Design,” in *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling ’20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 1–5. [Online]. Available: <https://dl.acm.org/doi/10.1145/3428029.3428059>
- [123] P. Rayavaram, A. Jagadeesha, S. Narain, and C. S. Lee, “Designing a Visual Cryptography Curriculum for K-12 Education,” in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, ser. SIGCSE 2023. New York, NY, USA: Association for Computing Machinery, Mar. 2023, p. 1319. [Online]. Available: <https://doi.org/10.1145/3545947.3576266>
- [124] L. Zhang, J. Nouri, and L. Rolandsson, “Progression Of Computational Thinking Skills In Swedish Compulsory Schools With Block-based Programming,” in *Proceedings of the Twenty-Second Australasian Computing Education Conference*, ser. ACE’20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 66–75. [Online]. Available: <https://doi.org/10.1145/3373165.3373173>
- [125] A. Milliken, V. Cateté, A. Limke, I. Gransbury, H. Chipman, Y. Dong, and T. Barnes, “Exploring and Influencing Teacher Grading for Block-based Programs through Rubrics and the GradeSnap

- Tool,” in *Proceedings of the 17th ACM Conference on International Computing Education Research*, ser. ICER 2021. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 101–114. [Online]. Available: <https://doi.org/10.1145/3446871.3469762>
- [126] N. Körber, “Anomaly detection in scratch assignments,” in *Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings*, ser. ICSE ’21. Virtual Event, Spain: IEEE Press, Nov. 2021, pp. 111–113. [Online]. Available: <https://doi.org/10.1109/ICSE-Companion52605.2021.00050>
- [127] D. Shepherd, P. Francis, D. Weintrop, D. Franklin, B. Li, and A. Afzal, “[engineering paper] an ide for easy programming of simple robotics tasks,” in *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2018, pp. 209–214.
- [128] D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin, “Blockly goes to work: Block-based programming for industrial robots,” in *2017 IEEE Blocks and Beyond Workshop (B&B)*, Oct. 2017, pp. 29–36.
- [129] N. Ritschel, V. Kovalenko, R. Holmes, R. Garcia, and D. C. Shepherd, “Comparing Block-Based Programming Models for Two-Armed Robots,” *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1630–1643, May 2022, conference Name: IEEE Transactions on Software Engineering.
- [130] H. L. Tan, C. C. Wong, J. N. M. Kho, and S. Hoh, “Simple Mobility Configurator: Block Programming for the Non-Experts,” in *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, Oct. 2008, pp. 1–4, iSSN: 2161-9654.
- [131] N. Ritschel, F. Fronchetti, R. Holmes, R. Garcia, and D. C. Shepherd, “Enabling end-users to implement larger block-based programs,” in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ser. ICSE ’22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 347–349. [Online]. Available: <https://doi.org/10.1145/3510454.3528644>
- [132] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, “Measuring program comprehension: A large-scale field study with professionals,” in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 584–584, ISSN: 1558-1225.
- [133] E. Wong, J. Yang, and L. Tan, “Autocomment: Mining question and answer sites for automatic comment generation,” in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013, pp. 562–567.
- [134] Y. Gao, H. Zhang, and C. Lyu, “EnCoSum: enhanced semantic features for multi-scale multi-modal source code summarization,” *Empirical Software Engineering*, vol. 28, no. 5, p. 126, 2023. [Online]. Available: <https://doi.org/10.1007/s10664-023-10384-x>
- [135] R. A. Boyd and S. E. Barbosa, “Reinforcement learning for all: An implementation using unreal engine blueprint,” in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2017, pp. 787–792.
- [136] P. Gestwicki, “Unreal engine 4 for computer scientists,” *J. Comput. Sci. Coll.*, vol. 35, no. 5, p. 109–110, oct 2019.

- [137] D. Asenov and P. Muller, “Envision: A fast and flexible visual code editor with fluid interactions (overview),” in *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2014, pp. 9–12, ISSN: 1943-6106. [Online]. Available: <https://ieeexplore.ieee.org/document/6883014>
- [138] S. Norhudha Sarif, S. Idris, and A. M. Zin, “The design of blocks integration tool to support end-user programming,” in *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*, Jul. 2011, pp. 1–5, iSSN: 2155-6830. [Online]. Available: <https://ieeexplore.ieee.org/document/6021657>
- [139] C. Kyfonidis, N. Moumoutzis, and S. Christodoulakis, “Block-C: A block-based programming teaching tool to facilitate introductory C programming courses,” in *2017 IEEE Global Engineering Education Conference (EDUCON)*, Apr. 2017, pp. 570–579, iSSN: 2165-9567.
- [140] Y. Lin and D. Weintrop, “The landscape of block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming,” *Journal of Computer Languages*, vol. 67, p. 101075, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S259011842100054X>
- [141] “Blockpy,” <https://think.cs.vt.edu/blockpy/blockpy>, Accessed on February 18, 2023. [Online]. Available: <https://think.cs.vt.edu/blockpy/blockpy>
- [142] C. M. Leon, E. Aizpurua, and S. van der Valk, “Agree or disagree: Does it matter which comes first? an examination of scale direction effects in a multi-device online survey,” *Field Methods*, vol. 34, no. 2, pp. 125–142, 2022. [Online]. Available: <https://doi.org/10.1177/1525822X211012259>
- [143] F. Funke, U.-D. Reips, and R. K. Thomas, “Sliders for the smart: Type of rating scale on the web interacts with educational level,” *Social Science Computer Review*, vol. 29, no. 2, pp. 221–231, 2011. [Online]. Available: <https://doi.org/10.1177/0894439310376896>
- [144] Ø. Langsrud, “ANOVA for unbalanced data: Use Type II instead of Type III sums of squares,” *Statistics and Computing*, vol. 13, no. 2, pp. 163–167, Apr. 2003. [Online]. Available: <https://doi.org/10.1023/A:1023260610025>
- [145] C. C. Gramazio, D. H. Laidlaw, and K. B. Schloss, “Colorgorical: Creating discriminable and preferable color palettes for information visualization,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 521–530, Jan. 2017, conference Name: IEEE Transactions on Visualization and Computer Graphics.
- [146] L. Bartram, A. Patra, and M. Stone, “Affective Color in Visualization,” in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. Denver Colorado USA: ACM, May 2017, pp. 1364–1374. [Online]. Available: <https://dl.acm.org/doi/10.1145/3025453.3026041>
- [147] M. Rollins, “Beginning lego mindstorms ev3,” in *Beginning LEGO MINDSTORMS EV3*. A Press, Berkeley, CA, 2014, 2014.
- [148] L. P. Flannery, B. Silverman, E. R. Kazakoff, M. U. Bers, P. Bontá, and M. Resnick, “Designing scratchjr: Support for early childhood learning through computer programming,” in *Proceedings of the 12th International Conference on Interaction Design and Children*, ser. IDC ’13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1–10. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/2485760.2485785>



- [149] H. T. Tran, H. H. Dang, K. N. Do, T. D. Tran, and V. Nguyen, “An interactive Web-based IDE towards teaching and learning in programming courses,” in *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, Aug. 2013, pp. 439–444. [Online]. Available: <https://ieeexplore.ieee.org/document/6654478>
- [150] W. W. Gaver, “Technology affordances,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '91. New York, NY, USA: Association for Computing Machinery, 1991, p. 79–84. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/108844.108856>
- [151] D. Norman, *The design of everyday things: Revised and expanded edition*. Basic books, 2013.
- [152] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, “Measuring program comprehension: A large-scale field study with professionals,” *IEEE Transactions on Software Engineering*, vol. 44, no. 10, pp. 951–976, 2018.
- [153] J. B. Arnold, “Tufte’s box plot,” [https://jrnold.github.io/ggthemes/reference/geom\\_tufteboxplot.html](https://jrnold.github.io/ggthemes/reference/geom_tufteboxplot.html), Accessed on March 22, 2024. [Online]. Available: [https://jrnold.github.io/ggthemes/reference/geom\\_tufteboxplot.html](https://jrnold.github.io/ggthemes/reference/geom_tufteboxplot.html)
- [154] R. Mason and G. Cooper, “Distractions in programming environments,” in *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*, 2013, pp. 23–30.
- [155] N. Peitek, A. Bergum, M. Rekrut, J. Mucke, M. Nadig, C. Parnin, J. Siegmund, and S. Apel, “Correlates of programmer efficacy and their link to experience: a combined eeg and eye-tracking study,” in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 120–131. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/3540250.3549084>
- [156] A. G. de Siqueira, P. Feijóo-García, S. Carnell, E. Palmeira, and A. Maxim, “fableblocks: Toward mitigating programming anxiety with storytelling-based tangible block programming environments,” in *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2022, pp. 1–4.
- [157] A. Louis, S. K. Dash, E. T. Barr, M. D. Ernst, and C. Sutton, “Where should i comment my code? a dataset and model for predicting locations that need comments,” in *2020 IEEE/ACM 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2020, pp. 21–24.
- [158] N. Rao, J. Tsay, M. Hirzel, and V. J. Hellendoorn, “Comments on comments: Where code review and documentation meet,” in *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 2022, pp. 18–22.
- [159] E. Thiselton and C. Treude, “Enhancing python compiler error messages via stack,” in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–12.
- [160] B. A. Becker, “An effective approach to enhancing compiler error messages,” in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ser. SIGCSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 126–131. [Online]. Available: <https://doi-org.ezproxy.library.unlv.edu/10.1145/2839509.2844584>

# Curriculum Vitae

Graduate College  
University of Nevada, Las Vegas

Alex Hoffman  
alexzh1907@gmail.com

## Degrees:

Bachelor of Science in Computer Science 2000, Hardin Simmons University

Master of Business Administration 2018, University of Nevada Las Vegas

Thesis Title: An Empirical Investigation into the Transitional Friction of Block-Based Programming Languages

## Thesis Examination Committee:

Chairperson, Dr. Andreas Stefik, Ph.D.

Committee Member, Dr. Hal Berghel, Ph.D.

Committee Member, Dr. Laxmi Gewali, Ph.D.

Committee Member, Dr. Fatma Nasoz, Ph.D.

Graduate Faculty Representative, Dr. Gregory Moody, Ph.D.

## RESEARCH AREAS

---

Data Science & Computer Science Education, Human Factors in Computing, Accessible Computing

## EDUCATION

---

IN PROGRESS PhD, Computer Science, University of Nevada, Las Vegas, Las Vegas, NV, Anticipated May 2023

2018 MBA, University of Nevada, Las Vegas, Las Vegas, NV

2000 BS, Computer Science, Math Minor, Hardin-Simmons University, Abilene, TX

## PROFESSIONAL EXPERIENCE

---

University of Nevada - Las Vegas – Las Vegas, NV 2017 – present

Lecturer, Howard Hughes Engineering, Computer Science (Graduate Faculty Status) 2020 – present

- Developed courses for new data science programs (graduate & undergraduate) including textbook selection and all course material creation
- Trained other instructors in teaching data science programming courses I developed and mentored them throughout the semester
- Teaching in-person and online (synchronous & asynchronous)
- Student assessment, development, and mentoring

Graduate Assistant, Troesh Center for Entrepreneurship and Innovation 2017 – 2023

- Development, maintenance, and security of the websites
- NSF I-Corps outreach
- Conference planning and hosting

Camp Manager, GenCyber Camp 2019, 2021

Program is sponsored by the NSA and NSF to address shortage of cybersecurity professionals

- Taught cyber security concepts and skills to high school students in a summer program
- Taught students how to design and present a research poster at a conference
- Oversaw all the student workers, assigned tasks, and designed/coordinated activities

FSI - President 2015 – present

FSI is a technology consulting firm and incubator. Areas of expertise include accelerating early-stage start-ups with software and web development, cybersecurity excellence, agile & lean process, and globalization. Clients span North America, Europe, and Asia.

Kalodio – Las Vegas, NV – Founder & CEO 2015 – 2016

Created the company, managed releases, coordinated with team members, and wrote all the server code utilizing TDD and automated deployment. Kalodio enabled people who were unable to (or don't want to) leave their home to work out with a personal trainer via live, two-way video over smartphones and tablets.

Wiper – New York, NY – Co-Founder & VP, Product & Engineering 2014 – 2015

Co-founded the company and was fully responsible for all product management, UI/UX design, engineering, cybersecurity, and technical operations. Wiper was built to enable users to privately send

messages and then “wipe” them away. Within 9 months, Wiper grew to 10MM+ registered users and was the #1 overall app in 24 countries.

Unison Technologies – New York, NY – VP, Product Engineering 2013 – 2015  
Established this new position with full responsibility for all engineering, cybersecurity, technical operations product management, and UI/UX design for this entrepreneurial start-up with a budget of \$2M.

QuickOffice – Dallas, TX | Kharkov, Ukraine | Pune, India 2006 – 2013  
Director of Engineering - Kharkov, Ukraine 2009 - 2013  
Established this new position with full responsibility for all engineering, IT & cybersecurity, and operations for the newly created iOS division (iPhone, iPad), and acted as the General Manager for the Ukraine-based division. Managed 10 direct reports (technical managers, IT/cybersecurity, project managers, HR, facilities)—125 total. Matrix managed 5 (UI/UX, product managers).  
Development Manager - Plano, TX 2007 – 2009  
ScrumMaster - Plano, TX 2006 – 2007  
Automation Engineer - Plano, TX 2006  
Nortel Networks – Richardson, TX – Member of Scientific Staff 2001 – 2006

## PUBLICATIONS

---

### PEER-REVIEWED JOURNAL ARTICLES

- Under Revision Hoffman, A., Stabler, H., Stefik, A., “Block Viz: An Empirical Comparison of Professionals' and Students' Assessment of Block-Based Programming Attributes” Empirical Software Engineering
- 2021 Hoffman, A. Et al. “Bountychain: Toward Decentralizing a Bug Bounty Program with Blockchain and IPFS,” International Journal of Networked and Distributed Computing. June 2021
- 2019 A. Hoffman and H. Berghel, “Moral hazards in cyber vulnerability markets,” Computer, vol. 52, no. 12, pp. 83–88, 2019.

### PEER-REVIEWED CONFERENCE PAPERS

- 2024 Stefik, A., Allee, W., Contreras, G., Kluthe, T., Hoffman, A., Blaser, B., Ladner, R., “Accessible to Whom? Bringing Accessibility to Blocks,” Conference: 2024 ACM SIGCSE, March 2024
- 2021 Austria, A., Park, C., Hoffman, A., Kim, Y., “Performance and Cost Analysis of Sia, a Blockchain-Based Storage Platform,” Conference: 2021 IEEE/ACIS 6th International Conference on Big Data, Cloud Computing, and Data Science (BCD), September 2021
- 2020 Hoffman, A., Becceril, E., Moreno, K., Kim, Y. “Decentralized Security Bounty Management on Blockchain and IPFS,” IEEE CCWC 2020, March 12, 2020.

## PRESENTATIONS

---

## ACADEMIC CONFERENCES

- 2020 “Decentralized Security Bounty Management on Blockchain and IPFS,” IEEE CCWC 2020. January, 2020.

## POSTERS

- 2023 Alex Hoffman. Polyglot Programming with Eye Tracking, UNLV College of Engineering, Las Vegas, NV.
- 2019 A Hoffman, E Becerril, K Moreno. Decentralized Security Bounty Management on Blockchain and IPFS Blockchain Day, UNLV College of Engineering, Las Vegas, NV.

## ADDITIONAL PRESENTATIONS

- 2019 Alex Hoffman, Eric Becerril, Kevin Moreno. Blockchain Day, UNLV College of Engineering, Las Vegas, NV.
- 2018 “Crypto as a Currency.” Speaker Opportunity/Challenge. UNLV, Las Vegas, NV.
- 2009 Invited Guest Speaker. iPhone Dev Camp. Kiev, Ukraine

## COURSES TAUGHT

---

2024 Spring	DA 621	Programming for Data Analytics I	UNLV
2024 Spring	DA 622	Programming for Data Analytics II	UNLV
2024 Spring	DA 651	Managing Big Data and Web Databases	UNLV
2023 Fall	DA 621	Programming for Data Analytics I	UNLV
2023 Fall	DA 622	Programming for Data Analytics II	UNLV
2023 Fall	DA 651	Managing Big Data and Web Databases	UNLV
2023 Spring	CS 138	Intro to Programming for Data Science I in Python	UNLV
2022 Spring	CS 138x	Intro to Programming in Python	UNLV
2021 Fall	CS 135	Computer Science I (in Python)	UNLV
2020 Fall	CS 140	Computing Languages: Python	UNLV

## SERVICE

---

- 2023–Present EPIQ Steering Committee (Experience Programming in Quorum)
- 2020–2022 GCEC Event Coordination (Global Consortium of Entrepreneurship Centers)
- 2005–2017 Board of Young Associates, Hardin-Simmons University, Abilene, TX (service alumni board)

## ACADEMIC WORKSHOPS AND CONFERENCES ATTENDED

---

### CONFERENCES

- 2022 Troesh Research Conference, Las Vegas, NV, Nov 2022

- 2022 GCEC: Global Consortium of Entrepreneurship Centers Conference, Las Vegas, NV, Oct 2022
- 2020 GCEC: Global Consortium of Entrepreneurship Centers Conference, Las Vegas, NV, Oct 2020
- 2020 USC Election Cybersecurity Initiative, Las Vegas, NV, Feb 2020
- 2019 B-Sides, Las Vegas, NV, Aug 2019
- 2019 DEF CON, Las Vegas, NV, Aug 2019
- 2019 Colloquium for Information Systems Security Education, Las Vegas, NV, Jun 2019
- 2019 Blockchain Day, UNLV College of Engineering, Las Vegas, NV, May 2019

## WORKSHOPS

- 2021 “Resume or Curriculum Vitae: How to Create Them and When to Use Them” UNLV Office of Online Education, Las Vegas, NV
- 2021 “Inclusive Teaching Practices” UNLV Office of Online Education, Las Vegas, NV
- 2021 “Teaching with Business Cases Online for Creativity, Collaboration, & Cohesion” UNLV Office of Online Education, Las Vegas, NV
- 2020 “Best Practices in Online Teaching” UNLV Office of Online Education, Las Vegas, NV
- 2020 “NASA/DRI Cybersecurity Bootcamp.” Las Vegas, NV, Jan 13-17, 2020.
- 2019 “How to Design Interactive Course Activities.” UNLV Office of Online Education, Las Vegas, NV.
- 2019 “Foundations of Accessibility.” UNLV Office of Accessibility Resources, Las Vegas, NV.
- 2019 “Malware Traffic Analysis.” B-Sides, Las Vegas, NV, Aug 6-7, 2019.
- 2018 “Blockchain Workshop.” UNLV Department of Computer Science, College of Engineering, Las Vegas, NV.

## AWARDS & DISTINCTIONS

---

Best Paper: “Decentralized Security Bounty Management on Blockchain and IPFS,” IEEE CCWC 2020  
 Awarded as Best Paper in the category of Cryptography and Information Security.

NASA/DRI Cybersecurity Bootcamp. Las Vegas, NV

Competitively selected (20% acceptance) to attend a workshop hosted by the Desert Research Institute and funded by NASA to teach cybersecurity for robotics, UAVs, IoT, and big data.

## PROFESSIONAL CERTIFICATIONS

---

- PMI Agile Certified Professional
- Certified ScrumMaster
- Certified Scrum Product Owner
- Certified Scrum Professional

## PROFESSIONAL ASSOCIATIONS & STUDENT GROUPS

---

Project Management Institute

Phi Kappa Phi Honor Society—Inducted in 2018

IEEE

National Cybersecurity Student Association 2019-2021

Layer Zero 2018-2021

Shad0w Synd1cate 2018-2021

Rebel Venture Fund 2017-2018

#### COMMUNITY SERVICE

---

2023 – present	City of Refuge Ministries, Ghana (K–12), Tema, Ghana
2016 – present	Three Square, Las Vegas, NV (Volunteer)