



# Empowering Novice Programmers: Redesigning Documentation for Effective Informal Learning

Journal of Educational Computing Research  
2025, Vol. 63(3) 695–727  
© The Author(s) 2025  
Article reuse guidelines:  
[sagepub.com/journals-permissions](https://sagepub.com/journals-permissions)  
DOI: [10.1177/07356331241311505](https://doi.org/10.1177/07356331241311505)  
[journals.sagepub.com/home/jec](https://journals.sagepub.com/home/jec)  


Icy (Yunyi) Zhang<sup>1</sup> , Yunqi (Ruby) Jia<sup>2</sup>,  
Xiaoxuan (Alicia) Cheng<sup>2</sup>, Ji Y. Son<sup>3</sup>, and James W. Stigler<sup>2</sup>

## Abstract

Although programming is often learned through formal instruction or self-paced tutorials, informal learning, for example, through publicly available online documentation, is also a significant resource for skill development among novices. However, many novices struggle to extract useful information from documentation. This work aims to answer two key questions: (1) What specific elements of documentation make it difficult for novice learners? (2) What are some design features that could enhance the effectiveness of documentation for novices? Study 1 aims to qualitatively explore the impact of three hypothesized design changes on novice programmers' learning experience: moving examples to the top of the documentation, ordering examples in increasing argument complexity, and displaying example outputs next to example codes. We propose these design changes should facilitate novices' learning from the documentation, enhancing its accessibility and effectiveness. Studies 2 and 3 evaluate the effectiveness of the redesigned documentation on novices' learning outcomes through a quantitative method. Our results demonstrate that these design changes help novices navigate and apply documentation more effectively.

<sup>1</sup>Department of Educational Psychology, University of Wisconsin-Madison, Madison, WI, USA

<sup>2</sup>Department of Psychology, University of California Los Angeles, Los Angeles, CA, USA

<sup>3</sup>Department of Psychology, California State University Los Angeles, Los Angeles, CA, USA

## Corresponding Author:

Icy Zhang, Department of Educational Psychology, University of Wisconsin-Madison, Madison, WI 90095-1563, USA.

Email: [icy.zhang@wisc.edu](mailto:icy.zhang@wisc.edu)

**Keywords**

programming documentation, novice programmer, schema, expertise, programming education, representational mapping

**Introduction**

Over the past decade, computer programming course enrollment has increased for both computer science (CS) and non-CS majors (Camp et al., 2015; Zweben & Bizot, 2021). However, computer programming is notoriously difficult for novices to learn, especially for novices from underrepresented backgrounds (Dasgupta & Hill, 2017; Soosai Raj et al., 2018). Despite its utility and rewarding career promises, students often experience college-level programming classes as painful, resulting in high dropout and failure rates (Piteira & Costa, 2013; Robins et al., 2003; Sorva, 2013; Tan et al., 2009). Efforts by researchers and instructors to address these issues have included creating better learning environments with simplified programming syntaxes (Gross & Powers, 2005; Mehmood et al., 2020; Peng et al., 2019; Sarkar et al., 2016) and developing more effective pedagogies (Eghterafi et al., 2022; Gross & Powers, 2005; McDowell et al., 2003) to support novice programmers.

However, many of these efforts focus on formal classroom environments and self-paced online tutorials, or on evaluating and revamping the syntax of programming languages (Perera et al., 2021). Although we certainly need to improve formal instruction and make programming syntax more accessible, a substantial amount of programming knowledge is acquired in less formal situations (Begel & Ko, 2019), for example, by “googling” or using AI tools for code assistance (e.g., ChatGPT, GitHub Copilot). Among the various resources learners encounter during informal learning, programming documentation stands out as one of the most frequently accessed and least studied. Unlike tutorials or forums, documentation is often designed with experienced users in mind, which can pose challenges for novice learners.

Our investigation focuses on programming documentation as a critical resource for informal learning especially for novices. We chose to focus on R, an open-source programming language for statistical computing and graphics (R Core Team, 2021), because it has a large novice user base (Shilane et al., 2024), particularly in introductory statistics courses at the college level. R’s relevance extends beyond statistics to interdisciplinary fields such as data science, where programming is increasingly seen as a necessary skill. The insights gained from studying R documentation could inform broader efforts to improve documentation practices across other widely used programming languages and enhance AI-driven tools that are trained on such documentation.

How might a student encounter documentation as they attempt tasks beyond the scope of their formal instruction? For example, a student who has learned a little about making basic data visualizations in R might search the internet or consult an AI assistant for help creating a boxplot, leading them to documentation for a commonly used

data visualization functions such as `geom_boxplot()` from the `ggplot2` package in R (Wickham, 2016). While documentation is useful to experienced users, it can be challenging for novices to navigate without guidance. Our study aims to identify features that make documentation more beginner-friendly, such as reordering content to introduce simple examples earlier. Investigating the formatting, sequencing and presentation of examples in documentation can move us closer to developing standards for beginner-friendly coding resources (Perera et al., 2021).

We do not argue that programming documentation should serve as a primary learning tool or be entirely rewritten for novices. Instead, we recognize that documentation is widely available, often included in search results or AI training sets, and regularly used by both experts and novices. By uncovering novices' strategies and challenges in using documentation, we aim to propose small changes in formatting and design that improve usability without negatively affecting expert users. This approach draws inspiration from the "curb cut effect" – modifications designed for one group, such as people in wheelchairs, often benefit the broader population, such as parents with strollers, travelers with luggage, and delivery workers.

The questions guiding this research are: (1) What makes documentation difficult for novice learners? (2) What are some design features that might help novice learners learn more effectively from documentation? Thus, the focus of the current paper is to investigate and validate the struggles encountered by novice programmers and to test design changes to alleviate these difficulties. These findings could shed light on the cognitive differences between novice and expert programmers, inform the design of future documentation, and foster inclusivity across diverse skill levels within the R coding communities. Although documentation practices vary across coding communities, this study focuses on the R documentation that is hosted on CRAN, a network of servers that store up-to-date versions of R code and documentation.

### *Documentation: Intended for Experts, Difficult for Novices*

The standard format for the documentation of R functions typically has five sections: descriptions, usage, arguments, details, and examples (see, e.g., [https://www.rdocumentation.org/packages/ggplot2/versions/3.3.6/topics/geom\\_boxplot](https://www.rdocumentation.org/packages/ggplot2/versions/3.3.6/topics/geom_boxplot)). The current design of documentation generally caters to the needs of experienced developers and programmers (Cogo et al., 2022). It is intended to effectively communicate with experts who are already familiar with the programming language (R in this case), the structure of a function, and often the package (or family of packages) that the function is part of. Experts, who have a lot of experience with documentation either as users or as authors, can efficiently locate and extract the information they need in documentation (Jeong et al., 2009). For example, an expert might quickly identify the code structure in the description or easily bypass general descriptions and skip directly to sections of the documentation that are most relevant to the task at hand.

While this structure is logical and comprehensive to experts, it can overwhelm novice programmers. According to cognitive load theory, people have limited cognitive

resources, and when learning complex content, they can experience cognitive overload – where the cognitive resources needed to process the information exceeds the learner’s capacity (Chandler & Sweller, 1996; Sweller, 2010). The content and structure of programming documentation can create an unnecessary cognitive load (Garner, 2002; Kalyuga et al., 1998), especially for novices who lack the prior knowledge needed to filter relevant information effectively. To support this argument, the literature has documented many challenges that novice learners experience when trying to learn programming (Denny et al., 2011; Jadud, 2005; Perera et al., 2021). For example, learners have difficulty learning the syntax of code (Denny et al., 2011), predicting code output, and identifying the correct order of commands (Perera et al., 2021).

Despite the common use of documentation by novice programmers, no study to our knowledge has examined their interaction with documentation or explored ways to redesign it based on theories in cognitive psychology to reduce cognitive load. Our study aims to address this gap to improve the usability of documentation for diverse learners, particularly for novices in learning R programming.

### ***Review of the Cognitive Psychology Literature: Why Documentation Works for Experts but Not Novices***

Cognitive psychology provides insight into why novices and experts experience documentation differently. First, experts possess structured and schematic knowledge, called schemas, within their domain of expertise; this allows them to navigate, search, and extract the information they need efficiently (Carbonell et al., 2014; Chi, 2011; Ericsson et al., 2018; Robins et al., 2003), ignoring what they deem as irrelevant and integrating multiple sources of information (see Fries et al., 2021 for a review). For example, an expert might know that they can skip directly to the end of the R documentation to locate the examples they need, without reading through the preceding sections (i.e., descriptions, usage, arguments, and details). In contrast, a novice may lack this schematic understanding, requiring more effort to navigate documentation and making them prone to feeling overwhelmed by irrelevant information.

Second, experts have extensive experience writing and running many lines of code; this enables them to infer causal relations between code and its output (Holyoak & Cheng, 2011). This knowledge of cause-effect relationships between the code and its output allows them to predict output (Tucker et al., 2024) and can be used to problem solve with code (Iwamoto et al., 2020). Novice programmers who do not have such accumulated experience struggle to predict code output (Lister et al., 2004) or to identify the code examples that match their goals (Iwamoto et al., 2020; Kashima, 2019).

Lastly, experts possess a schematic understanding of coding syntax that allows them to interconnect different functions and components of code (Crk et al., 2015). Experts can parse code into relevant and irrelevant parts and easily recognize structurally related examples (Jeffries, 1981; Wiedenbeck, 1985). This structural knowledge of the code enables experts to modify code chunks accurately for a new problem or task. For

example, an expert creating a data visualization might easily identify which parts of the code change features such as color or line thickness and which parts change the variable or dataset. Novices seem to have fragmented knowledge of individual pieces of code rather than meaningful schemas (Bosse & Gerosa, 2017; Gomes & Mendes, 2007; Luxton-Reilly, 2016; Robins, 2019) and often struggle with the syntax and structure of code (Perkins et al., 2013; Piteira & Costa, 2013).

In summary, experts seem to possess at least three cognitive schemas that facilitate their use of documentation: (1) a schematic understanding of documentation itself; (2) a causal understanding of the relationship between code and its output; and (3) a schematic understanding of the coding syntax. Novices, on the other hand, lack these schemas which leads to difficulties in navigating documentation, identifying relevant examples, and modifying code for a new purpose (Ichinco & Kelleher, 2015; Lister et al., 2004; Robins et al., 2003; Wiedenbeck et al., 1993). The lack of schemas provide an explanation for why novices might have trouble learning from current forms of documentation (Chandler & Sweller, 1996; Mow, 2008). Specifically, we predict that novices navigating current forms of documentation will have three difficulties: (1) navigating documentation; (2) identifying relevant examples; and (3) modifying examples appropriately. These predictions will be examined in Study 1.

### *Design Changes Hypothesized to Make Documentation More Accessible for Novices*

Presenting materials in an instructionally purposeful form can facilitate relational learning and aid schema formation (Clement et al., 1986). These formatting changes may act as scaffolding that helps novices build up the schematic understanding that experts possess. Although experts may not need such scaffolding, naive learners benefit from additional alignment and purposeful integration in the presentation of new ideas (Kalyuga et al., 1998; Perera et al., 2021). In this section, we present three hypotheses for design changes in programming documentation that aim at facilitating novices' learning that we will test in the current studies.

*Hypothesis 1. Reordering Documentation to Prioritize Examples.* We hypothesize that reordering the subsections of R documentation to move sections that are most useful to novices to the beginning (e.g., examples) and the sections least useful to novices to the end (e.g., usage and arguments) can increase learning. Examples are often the most helpful for novices yet they are typically located at the end. Novices, who lack a structured schema for navigating documentation, often fail to locate them. Sections on usage and arguments of a function, while important for experts, present a cognitive gauntlet that learners must pass through in order to arrive at the examples. This cognitive overload may generate frustration and confusion (Bakar et al., 2019; Garner, 2002). Reordering sections to prioritize examples may allow novices to allocate more of their cognitive resources to processing, learning from,

and applying the examples rather than struggling through irrelevant sections (Sweller, 1988).

*Hypothesis 2. Integrating Output with Example Code.* Our second hypothesis is that presenting the output alongside the corresponding example code will help novice programmers interpret and modify the example code more effectively for their tasks. Because novices lack a causal understanding of the relationship between the code and output, they need explicit scaffolding to build and make use of such knowledge. This design change can support cognitive processes such as comparison and mapping (Zhang et al., 2024), helping novices infer causal relationships and develop schemas for understanding code structure (Clement et al., 1986). By visually aligning the cause (code) and effect (output), novices can identify patterns and relationships that are critical for reasoning about programming tasks. We hypothesize that this code-and-output alignment will help novices modify code examples for new purposes.

*Hypothesis 3. Ordering Examples from Simple to Complex.* Lastly, we hypothesize that presenting examples in order of increasing complexity, based on the arguments used, will help novices learn incrementally. Organizing examples progressively creates opportunities to discover structural differences and similarities between examples. This design principle is supported by the representational mapping literature, which suggests that the order of instances that lead novices to notice structural patterns facilitates schema development (Kotovskiy & Gentner, 1996; Scheiter & Eitel, 2015; Zhang et al., 2024).

Incrementally ordering examples can also alleviate some of the cognitive load that would otherwise be spent searching for appropriate code comparisons. The incremental ordering of examples, combined with the side-by-side presentation of output provides novices with multiple opportunities to engage in comparison: (1) across example codes, (2) across outputs, and (3) between changes in the code and their corresponding changes in output. These comparison opportunities can promote deeper relational learning and develop a structural understanding of the topic (Gentner et al., 2021).

We summarize these schemas, behaviors, and design proposals in Table 1. By forefronting examples, aligning code with output, and ordering examples by complexity, these changes can potentially compensate for novices' lack of schemas, reduce their cognitive load, and provide scaffolded opportunities for developing cognitive schemas. These hypotheses will be tested in the studies that follow, providing insights into how documentation can be redesigned to better serve a broader range of learners.

## Current Studies

In the following studies, we aim to document how novices interact with documentation as it exists today and explore whether the proposed design changes might aid their

**Table 1.** Schemas, Behaviors, and Design Proposals.

Experts' Schemas	Novices' Predicted Behavior (What Happens Without the Schema)	Proposed Design Feature (How to Make up for the Lack of Schema)
Schema of R documentation	Difficulty navigating the documentation and even finding the example section	Moving the example section to the top of the documentation
Causal connections between code and output	Difficulty identifying the most helpful example from the documentation	Presenting output alongside the example code
Causal connections between different parts of the code	Difficulty adapting or modifying examples from the documentation to solve new problems	Reordering the examples incrementally from simple to complex

effective use of documentation. We took the documentation for the *geom\_boxplot* function as an example and created a redesigned version implementing the three design changes mentioned above.

In Study 1, individual novice learners engaged in think-alouds as they navigated either the original or redesigned documentation. These sessions provided qualitative insights into how novices interact with documentation and the challenges they encounter. In Studies 2 and 3, we examine experimentally whether these design features would allow novices to learn more from documentation. In all three studies, participants in a statistics course who were just starting to learn R were randomly assigned to receive either the original or redesigned documentation. Their ability to use the documentation was tested through a series of coding tasks. In each, they were shown a graph and asked to determine the code to generate it.

We asked novices to generate a series of boxplots because it is a cognitively engaging task that is accessible to novices. Code for making such data visualizations is also cognitively interesting because it requires learners to map parts of the code to different features of a graph. Making data visualizations also highlights the potential benefits of presenting code and output together and encourages schema formation for the syntactic structures shared by graphing functions in the *ggplot2* package (Wickham, 2016). Developing such schemas can help learners create novel visualizations using related functions in the *ggplot2* package. Participants in our study were asked to learn the coding syntax of *ggplot2* (Wickham, 2016), which they had not yet encountered in their introductory statistics class. They were more familiar with the coding syntax of another package for making visualizations, *ggformula* (Kaplan & Pruim, 2022).

Through these studies, our goal is to provide evidence for the effectiveness of the proposed design changes and their impact on the usability of documentation for novice learners. By investigating novice behaviors and learning outcomes, this research seeks to bridge the gap between existing documentation formats and the needs of novice programmers.

## Study I

### Method

**Participants.** The participants were 17 students (seven males, 10 females) from a non-competitive regional public university recruited from introductory statistics courses where they were learning R (their class used a textbook from Course-Kata.org). One additional participant was excluded from the analysis because they did not complete the entire interview. The participants were recruited about halfway through the semester (Fall 2021), at a point where they had some familiarity with R but were still in the early stages of learning. Participants were compensated, according to their own preference, with either extra credit in their statistics class or a \$20 gift card (only one student chose the latter). All participants gave consent before participating in the study. The study was approved by the university's Institutional Review Board (IRB).

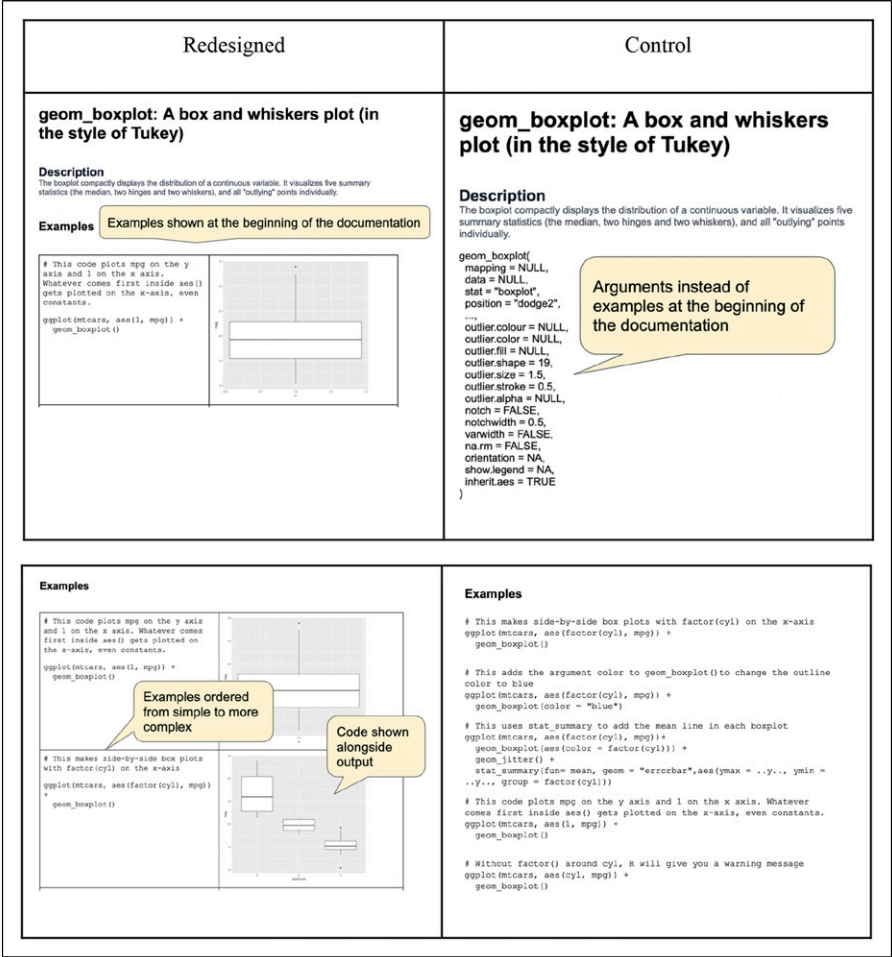
**Design.** The study employed a between-subjects design. Participants were randomly assigned to one of two conditions: the **control group** ( $n = 9$ ) where they received the original, unmodified documentation from CRAN or the **redesigned group** ( $n = 8$ ) where they received documentation modified with design changes hypothesized to improve usability for novices. In both conditions, participants were asked to use the documentation to recreate 10 data visualizations.

Two trained interviewers each interviewed half of the participants from both conditions. The interview sessions were conducted either in-person ( $n = 10$ ) or online via Zoom ( $n = 7$ ), based on the availability of the interviewer and interviewee. Each interview ran between 60 and 80 minutes.

**Materials.** Participants engaged in an online learning task in one browser tab and had access to R documentation on a different browser tab. They were allowed to flip between the task and the documentation and permitted to copy/paste from the documentation during the learning task.

**Documentation.** There were two versions of the documentation for the *geom\_boxplot* function (available at [https://osf.io/2az8m/?view\\_only=42561813de7c496cbe520e0c62a2a89f](https://osf.io/2az8m/?view_only=42561813de7c496cbe520e0c62a2a89f)). The control version was similar to the original *geom\_boxplot* documentation available on the R documentation website (<https://www.rdocumentation.org/>). The redesigned version was modified with three design changes: (1) sections less likely to be used by novices were moved to the end while examples were moved to the beginning of the page; (2) each snippet of example code was accompanied by the graph it would create; and (3) the examples were ordered from simplest to most complex. The redesigned version of the documentation was more similar to the ggplot2 reference for “geom\_boxplot” (available here: [https://ggplot2.tidyverse.org/reference/geom\\_boxplot.html](https://ggplot2.tidyverse.org/reference/geom_boxplot.html)) than the basic R documentation for the same function. Figure 1 illustrates the differences between the redesigned documentation and the control version.





**Figure 1.** Comparison between redesigned and control versions of the documentation.

**Learning Task.** Students were presented with a series of 11 questions designed to engage them in learning some new code. The questions started with a warm-up question that included code students learned in class to acclimate students to the think-aloud protocol. This was then followed by 10 additional questions, all presented as a Qualtrics survey (Qualtrics, 2022; see Online Appendix A for the full set of questions). The learning task asked participants to write code (seven questions), explain some part of the code (one question), or choose the appropriate code (two multiple-choice questions) to create a target graph.

Even though students were only provided with documentation about the function `geom_boxplot`, there were two types of plots involved in the learning tasks: boxplots

(seven questions) and violin plots (three questions). Students were told that a violin plot is a type of data visualization similar to a boxplot, which is created using the function *geom\_violin*. They were instructed to use the *geom\_violin* function the same way as they would use the *geom\_boxplot* function. The violin plot questions provided a way to measure whether students could transfer their understanding of *geom\_boxplot*, based on the documentation, to create violin plots. Students were not able to run code during the interview to see if it would work.

### Procedure

**In-Person Interviews.** Each of the in-person participants in the study ( $n = 10$ ) sat in front of a desktop computer with two browser tabs open: one displaying the learning task and the other showing the R documentation for *geom\_boxplot*. Even though participants were in-person, Zoom was used to record the screen and sound during the interview. Participants were introduced to the goal of the study and the think-aloud technique. They were instructed to attempt each question in the learning task and verbalize their thoughts in as much detail as possible. (The full version of the interview script is included in [Online Appendix C](#)).

After answering the warm-up question, students were given a brief description of the *ggplot2* package, the new package they were going to learn about. Participants were informed that they would have access to the documentation for a few *ggplot2* functions as they attempted to answer 10 questions about how to create graphs with the new functions. Before participants started to answer questions, the interviewers informed them that they were allowed to directly copy and paste code from the examples on the documentation page if they wanted.

**Online Interviews.** Because the in-person interviews were recorded via Zoom, the fully online interviews ( $n = 7$ ) were almost identical. The only difference was that at the start of the online interviews, students joined a meeting room and shared their screens. They were given links and instructed to set up their screens with the two tabs (one with the questions, the other with the documentation). From that point forward, the interviews followed the same procedure as in the in-person sessions.

**Coding and Analysis.** Two trained researchers transcribed the 17 interviews. Then, they analyzed the transcripts and participants' typed responses to questions (multiple choice, free response, and coding) during the interview. The coding of transcripts and responses focused on how novices: (1) navigated the documentation, (2) selected the most appropriate example(s), and (3) modified the code. For each question that required participants to write their own code, we identified in advance the examples that would be most relevant to the task and assessed whether or not students started with these examples (e.g., either by copying and pasting them or by studying them more closely).

In addition to coding task performance, the same two researchers also coded for students' emotional responses from their think-aloud responses when they approached the three coding questions that involved violin plots, a novel type of plot they had not

learned before. Emotional responses, such as expressions of fear, frustration, or confidence, were coded to evaluate how participants reacted to unfamiliar tasks and whether their emotional states changed throughout the process.

## Results

When reporting, we will refer to specific participants by first indicating the condition they were in (C for Control and R for Redesigned) and then an initial to represent their first name. The initials were based on fake names that we gave to each participant. For example, a participant in the Control condition assigned a name of Gill will be referred to as C\_G.

*Navigating Documentation.* All participants, regardless of condition, chose to read the documentation from the top to the bottom. Participants in the control condition read the list of arguments before they got to the examples due to the way the documentation was designed. Despite the fact that the list of arguments was not helpful for their tasks, control-condition participants dutifully started reading them when they first approached the documentation.

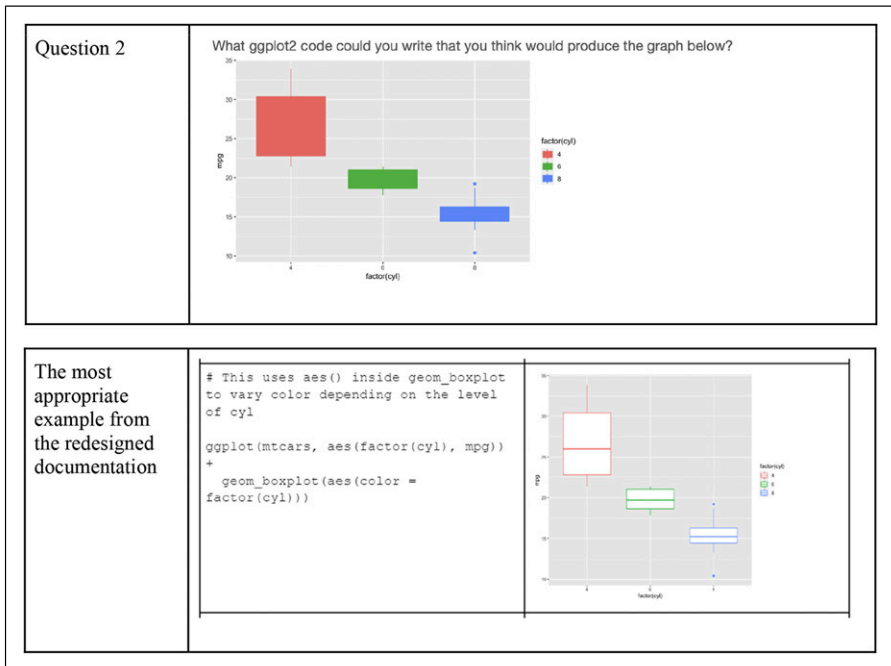
All eight participants in the redesigned condition navigated to the examples relatively quickly because they were presented first. Of the nine participants in the control condition, six eventually navigated to the examples on their own, whereas three students had to be prompted by the interviewer after spending a long time on the parts before the examples in order to find the examples.

Specifically, Participant C\_G was an interesting example to demonstrate the challenges novices face in navigating documentation. She never consulted the example part of the documentation when engaged in the learning task despite the interviewer prompting her to do so. She later revealed that she did not know what documentation was, so she may have even doubted whether consulting it would be useful. Upon arriving at the documentation, she started from the top, skimmed through the long list of arguments, stopped at the examples, and did not read them. She then went back to the survey and answered the first question incorrectly, which asked, "Which line of *ggplot2* code was used to produce the graph below? Select all the options that could have generated this graph." When this participant attempted the second question, she visited the documentation for the second time but only consulted the long list of arguments. She copied the argument "stat" from this section and attempted to invent her own syntax. This participant only found the examples helpful when she went back to the documentation a third time. The experience of this participant from the control condition, although somewhat extreme, demonstrated how difficult it can be for novices who did not possess a schematic understanding of the documentation to navigate the documentation itself. Although we assumed that novices would find examples more helpful than other parts of the documentation, for this participant, even determining whether examples versus arguments would be helpful was a challenge. It is likely that Participant C\_G

were overwhelmed by the description and the long list of arguments, leaving little cognitive resources for her to process the example codes.

**Searching for the Most Appropriate Example.** When first asked to write code to re-create a particular graph (Figure 2), 13 out of the 17 participants started by copying and pasting code from the documentation rather than by writing their own code. A common strategy for using documentation seems to be to identify some example code to start with, then revise the code as necessary to fit the situation. A key to success, however, is to find the most relevant example code to start with. Seven out of eight participants in the redesigned condition were able to find the most appropriate example from the documentation for this question, compared to only two out of nine participants in the control condition were able to do so.

Across the seven questions that required participants to write code, participants in the redesigned group zeroed in on the most appropriate example an average of 5.86 times ( $SD = 1.86$ ), whereas those in the control group did so less frequently ( $M = 3.57$ ,  $SD = 1.72$ ). Among participants who were able to find the most appropriate example in the documentation, participants in the redesigned group, across all questions, found the most appropriate example more quickly on average ( $M =$



**Figure 2.** The first question that asked participants to write code (question 2) and the most appropriate example from the documentation.

57.6 seconds,  $SD = 43.6$ ), than did those in the control group ( $M = 110.0$  seconds,  $SD = 26.9$ ). Because the control version of the documentation did not have plots next to the examples, they seemed to encounter more barriers as they attempted to identify a helpful example. For instance, when tasked with making a horizontal boxplot, participant C\_F failed to locate the appropriate example. As she skimmed through the documentation to understand what each piece of example code was doing, she expressed her confusion, “I’m just not sure which one exactly I should use in order to make them horizontal.”

All participants engaged in some copying and pasting, which they were encouraged by the interviewers to do at the beginning of the session. Because the redesigned documentation provided output graphs that could be mapped to the example code, we were interested to know whether students took advantage of this feature. We found that all participants in the redesigned condition consistently mapped between the graph they were asked to make in the learning task (e.g., horizontal boxplot) and the example plots in the documentation. These participants consistently searched through the example plots, flipped back and forth between the documentation and the survey question. It is likely that participants in the redesigned group benefited from the structure of the redesigned documentation such that they can more easily compare the graph they were asked to make with the examples in the documentation.

*Modifying an Example for the Current Purpose.* Once participants identified an appropriate example, the two versions of documentation did not seem to affect their ability to modify the example. Participants in both conditions commonly struggled to distinguish highly similar arguments such as color and fill. In *ggplot2*, the argument *fill* is used to fill in the boxplot with an interior color while the argument *color* is used to change the color of the outline. In Question 2, participants were tasked with making side-by-side boxplots *filled* with different colors according to a variable, but the most relevant example in the documentation *outlined* the boxplots in different colors according to a variable. Participants needed to change the color argument in the example to fill, but even participants who identified the most appropriate example (i.e., seven in Redesigned and two in Control) failed to change the argument from color to fill appropriately. For example, participant C\_B, ultimately included both fill and color arguments in her response while saying, “Color, yes, but also, fill too...because color just outlines it, but fill, like you know, fills it in.” The arguments fill and color may also be examples of arguments that are relatively easy to fix in more authentic situations when students have the opportunity to run code, a feature not available in this study.

There were a few cases when the visualizations in the redesigned documentation facilitated a clearer understanding of these confusable arguments. For example, to change the color of an outline according to a variable, students should set the color to vary by the name of the variable (e.g., `color = condition`), instead of setting it to a specific color (e.g., `color = “blue”`). Participant R\_C in the redesigned condition initially had thought he should include `color = “blue”`, but after he referred to the documentation and saw the examples colored according to a variable had `color =`

condition, he said in surprise, “Wow, that’s cool. It’s crazy”. This moment of discovery allowed him to make his eventual correct decision to use `color = condition`. The presence of example plots demonstrating variable-based coloring seemed to provide a direct and intuitive explanation that supported his understanding.

In summary, the redesigned documentation facilitated novices’ search for the most appropriate example, as evidenced by their higher success rates in locating examples and faster performance compared to the control group. However, once participants were able to find the correct example, there were few differences in how participants used the control and redesigned documentation for modifying the example.

*Emotional Reactions to New Code and Data Visualizations.* Our analysis of expert and novice cognition predicted differences in three types of behaviors - navigation, selection of examples, and modification of examples. However, the study revealed additional differences in emotional reactions—such as fear, confusion, and confidence—that were not anticipated by our analysis. Through our interviews, we saw that the differences between the two types of documentation had a tangible impact on the emotions experienced by the novices in our sample.

These emotional reactions were most pronounced when students were asked to write code to create violin plots in the last three questions of the learning task. While all participants had already learned about boxplots in their statistics class, violin plots were new to them. Because of this, students had to learn not only some new R code but also a new type of data visualization. Since participants were provided only with *geom\_boxplot* documentation and not *geom\_violin* documentation, this task tested their ability to transfer knowledge from boxplots to violin plots.

Initially, participants across both conditions expressed negative emotions. Specifically, five participants in each group reacted negatively when encountering violin plots (e.g., “No! There is nothing about [violin plots in the documentation!]”).

Despite these initial reactions, the redesigned documentation helped mitigate some participants’ negative emotions by supporting their efforts to transfer their knowledge about boxplots to violin plots. Four participants in the redesigned condition transitioned from feeling confused to feeling confident after engaging with the documentation compared to only two in the control group. For instance, participant R\_B, looking at the redesigned documentation, said, “Oh, so [the *geom\_boxplot* documentation] just stops there because we’ve been doing boxplots but now we are doing a violin plot...” After reflecting on the examples, he realized: “I would do the same thing except instead of putting boxplot, I would put violin.”

Several participants felt and *stayed* scared and frustrated about the new plot (3 control, 1 redesigned). Their persisting negative emotions throughout the task was revealed through their verbal expressions. For example, participant C\_A in the control condition disclosed her struggle upon seeing the violin graph: “Oh no, [the violin plot] looks like a ghost.” After referring to the documentation, she expressed her frustration: “No! There is nothing about [violin plots!]” This participant was overwhelmed and was unable to finish writing the code for the last question.

Notably, four out of the five participants who shared positive reactions to the documentation overall at the end of the interview came from the redesigned condition.

## *Discussion*

In Study 1, we applied a cognitive analysis approach to redesign R documentation for novice programmers. Our semi-structured think-aloud interviews provided qualitative data, revealing how novice programmers navigated and interpreted the standard and redesigned documentation to learn about novel R functions and apply novel R functions to solve programming questions. Students' responses provided a valuable look both at the cognitive and emotional aspects of their learning experience: what they were thinking and what they were feeling as they worked to interpret documentation.

Novices mostly started reading documentation from the beginning and did not exhibit the navigational and searching behaviors that experts would engage in. Because the most helpful examples were up front in the redesigned documentation, this novice behavior was not a hindrance to students in the redesigned condition. The redesigned documentation, particularly the output that was linked to the code examples, supported novices' search for the most helpful examples and helped them work through negative emotional responses as they approached a novel coding task. It is possible that the redesigned documentation, by decreasing the cognitive demand, increased participants' self-efficacy and sense of competence in the learning process, pointing to an increasing future direction that explores the affective aspects of learning that accompanies cognitive changes.

It is interesting that we did not, as we had hypothesized, find much difference in how novices modified example code between the redesigned and the control documentation. This could be due to the fact that only a subset of participants—primarily from the redesigned condition—identified the most relevant examples in the first place. The lack of differences in code modification may also reflect the study's relatively small sample size, which is a limitation of Study 1.

Another possibility is that the redesigned documentation helps find a relevant example, but it does not necessarily help learners figure out how to modify the code. In some cases, modification of code might be easier than the search if the only thing learners need to modify is the dataset or variable name. In other cases, learners might need more scaffolding to grasp how to distinguish between and modify similar arguments like fill and color.

Although these interviews offered valuable insight into how novice programmers use documentation, the qualitative nature of the interview and the small sample size do not allow us to draw causal conclusions about the advantages of the redesigned documentation.

## **Study 2**

In the next two studies, we set out to examine the effect of the redesigned documentation on naive programmers' learning with a larger sample size and a quantitative

approach. Specifically, in Study 2, we used the same documentation materials but instead of asking students to engage in think-aloud, students attempted to answer 12 questions in a Qualtrics survey (similar to the questions in the learning task in Study 1), and we coded participants' ability to complete coding questions appropriately. The 12 questions drew on (1) boxplots, for which documentation was provided, and (2) violin plots, for which documentation was not provided. Because the code to create a violin plot is structurally the same as the code to create a boxplot, the questions for violin plots were designed as transfer questions: were students able to glean enough of the structure of these functions to figure out violin plots? We also collected some attitudinal measures: students' perceptions of question difficulty, their confidence in their ability (pre-test) and their answers (post-test), and their perceptions of the helpfulness of the documentation.

We hypothesized that by controlling for self-rated pretest confidence in their programming ability (a proxy for their prior experience in coding), students who used the Redesigned documentation would outperform students who used the Control documentation, both on tasks directly related to *geom\_boxplot* and on transfer tasks involving *geom\_violin*. In addition, because we saw some differences in emotional reactions in Study 1, we wanted to explore whether we could pick up, quantitatively, any differences in students' posttest attitudes between the Redesigned and Control conditions, including confidence and perceptions of difficulty.

## Method

**Participants.** Participants were 106 undergraduate students taking a 10-week introductory statistics course at a competitive public university. (Due to the COVID-19 pandemic, the course was taught online.) The racial/ethnic composition of the sample reflected the general demographics of the university: 33.0% White, 43.4% Asian, 4.7% African American, and 18.9% Hispanic or Latino. Students volunteered to participate in the study for extra credit toward their final course grade and did not receive any other form of compensation. The institutional review board at the university approved the study. Consent was obtained from participants online.

**Design & Procedure.** Students were emailed by their instructor with an invitation to participate in the study near the middle of the 10-week course. By that point in class, they had learned to make a variety of visualizations (e.g., boxplots, histograms) using the *ggformula* R package. Although this sample of students was drawn from a university different from Study 1, the statistics courses that the students were enrolled in used the same textbook that incorporated learning of R (see [CourseKata.org](https://www.coursekata.org)). Students who wished to participate completed a Qualtrics survey that included a pre-test survey, learning tasks, and a post-task attitudinal measures.

In the pre-test survey, participants were asked to report their confidence that they could master the content of their introductory statistics course. Participants were then introduced to a new package, *ggplot2*, for creating graphs in R, after which they were



randomly assigned to access one of the two versions of R documentation (control or redesigned) used in Study 1. Students were tasked with using documentation of the `geom_boxplot` function to respond to 12 questions, including nine coding exercises, two multiple-choice questions, and one free-response question. As students worked through these questions, once they submitted a response, they could not revisit previous questions to revise their answers.

After answering the 12 questions, participants were asked to rate the difficulty of the coding questions, their confidence in their answers, and the helpfulness of the documentation using 7-point Likert scales.

### *Materials & Measures*

**Documentation.** The same two versions of R documentation used in Study 1—the redesigned version and the control version—were used for this study.

**Confidence.** At the beginning of the study, students rated their agreement on a 7-point Likert scale with five statements about their confidence in their ability to learn and perform well in the course: “I feel confident in my ability to learn this material”; “I am capable of learning the materials in this course”; “I am able to achieve my goals in this course”; “I feel able to meet the challenge of performing well in this course”; “I feel confident in getting an A in this course.” Students’ confidence was summarized by summing up their responses to each statement to create a composite score, with higher values indicating more confidence (max = 35).

**Learning Task.** Of the 12 graphing questions in the learning task, nine were about boxplots and three were transfer questions about violin plots. The first 10 questions were the same as those used in Study 1. Two additional coding exercises (Questions 11–12) were added for Study 2. [Online Appendix A](#) shows the complete list of 12 questions.

The multiple-choice and free-response questions were scored based on correctness, each worth one point. Each coding exercise was scored according to a predetermined grading rubric that awarded points for using the correct data set, functions, arguments, and variables. One point was awarded for each component of the task, and each question could earn up to five to eight points depending on the number of components in the question. Participants’ final score for each question was calculated by dividing their raw score by the maximum number of components for each question. For example, if a participant correctly wrote four components and the coding exercise had a total of five components, the participant would earn a score of 0.8 for that question. Thus participants could earn up to 12 points for the coding tasks. For multiple-choice questions, when there were multiple correct answers, a partial point of 0.5 was given to partially correct answers (e.g., selecting only one answer out of the two correct answers in the multiple-choice question).

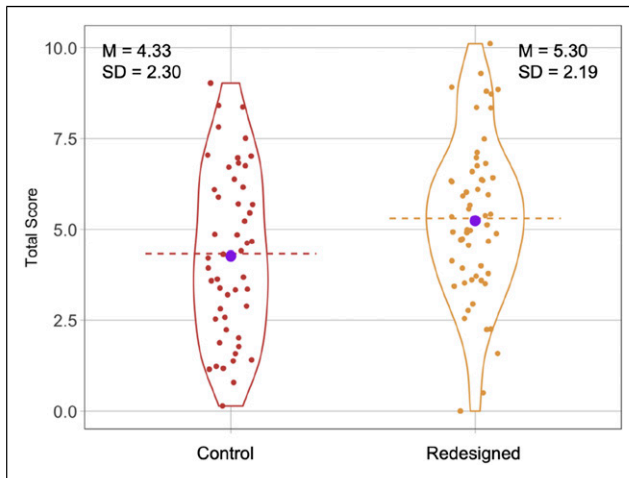
**Post-Survey Attitudinal Measures.** Students rated their perceived difficulty of the questions (“How hard were the coding questions (1 being not difficult at all)?”), their

confidence in their responses (“How confident are you about your answers (1 being not confident at all)?”), and their perceived helpfulness of the documentation (“How helpful was the documentation (1 being not helpful at all)?”), each on a 7-point scale. A full list of questions is included in [Online Appendix B](#).

## Results

The confidence ratings provided some insight into how they felt about coding at the beginning of the study, which was approximately five weeks into the class. The average confidence rating was 23.85 ( $SD = 6.78$ ) out of a maximum of 35. As a validity check of random assignment, we compared whether the two groups differed in their average confidence ratings prior to the intervention. An independent samples t-test showed that the confidence ratings did not differ significantly by condition ( $t(104) = 1.08$ ,  $\eta^2 = .01$ ,  $p = .285$ ).

**Performance on Graphing Questions.** Our main interest was in how the documentation would affect performance on the graphing questions (as shown in [Figure 3](#)). We performed an independent samples t-test and found that the experimental group performed significantly better than the control group ( $t(104) = 2.02$ ,  $\eta^2 = .04$ ,  $p = .046$ ). The effect of the condition remained significant when we controlled for students’ confidence before the intervention with an ANCOVA ( $t(103) = 2.37$ ,  $\eta_p^2 = .04$ ,  $p = .020$ ). In this analysis, confidence also significantly predicted performance ( $t(103) = 2.75$ ,  $\eta_p^2 = .04$ ,  $p = .007$ ).



**Figure 3.** Violin plot of participants’ scores on the coding questions by condition (study 2). The dashed line represents the mean and the purple dot the median.

*Transfer Questions.* Another question of interest from Study 1 was whether students in the redesigned group were more likely to transfer their understanding of *geom\_boxplot* to *geom\_violin* to create a new type of graph. In a multiple regression that included both condition and pre-survey confidence as predictors, we found significant effects of both condition ( $t(103) = 2.34, \eta_p^2 = .04, p = .021$ ), and confidence ( $t(103) = 2.92, \eta_p^2 = .08, p = .004$ ) on the violin graph questions, which were not directly addressed by the documentation.

*Attitudinal Measures.* Next, we investigated whether the two groups of participants differed on the three attitudinal measures at the end of the study (Table 2). An independent samples t-test showed no significant differences between the two groups in their ratings of the difficulty of the coding questions ( $t(103) = 1.68, \eta^2 = .03, p = .096$ ), their confidence in their answers ( $t(87) = .66, \eta^2 = .00, p = .513$ ), nor in the helpfulness of the documentation ( $t(98) = 1.60, \eta^2 = .03, p = .114$ ).

Study 2 Discussion

Consistent with the navigational behaviors exhibited by students in Study 1, the findings of Study 2 suggest that the redesigned documentation led to better performance on the learning task. These results provide evidence that design changes based on a cognitive analysis can facilitate novices’ successful use of documentation.

The redesigned documentation was intended to make up for three expert schemas that novices might lack (i.e., a schematic understanding of documentation, syntactic understanding of the coding language, and causal connections between code and output). We hypothesized that the corresponding design changes (i.e., forefronting examples, ordering example code by increasing complexity and providing output next to example code) would promote schema learning by supporting cognitive processes such as comparison, alignment, and mapping of the code as well as the resulting plots.

Study 3

In Study 2 we compared the standard documentation with a revised version that incorporated three major changes. Although we found a difference between the two conditions, it did not isolate which design features were most critical for novice

Table 2. Three Attitudinal Measures by Condition in Study 2.

Condition	Difficulty		Confidence		Helpfulness	
	M	SD	M	SD	M	SD
Control	6.26	0.96	2.20	1.27	3.19	1.51
Experimental	5.84	1.52	2.04	1.02	3.62	1.16

programmers’ performance improvement. Therefore, we cannot tell if all the changes were necessary, or if perhaps some of them were less relevant. We were especially interested in the provision of output next to examples. In Study 2, the output was presented right next to the examples. But it is not clear if the benefit was due simply to having the examples available, or if the layout itself was important.

The literature on representational mapping suggests that presenting two different types of representations (e.g., the code and the output) in a way that facilitates learners mapping between the two should benefit learning (Butcher, 2006; Martin et al., 2019; Scheiter & Eitel, 2015; Schmidt-Weigand et al., 2010; Thompson & Opfer, 2010). For example, integrating and making explicit the connections between diagrams and numerical representations supports students’ learning in Physics (van der Meij & de Jong, 2006).

Building on this, in Study 3 we started with the documentation from the redesigned condition but compared it to a degraded redesigned version in which the examples had to be brought up in a pop-up window when students asked for them. We hypothesize that the side-by-side presentation of code and output in the redesigned documentation would better support schema formation by facilitating comparison and alignment than the degraded redesigned condition where there was no such immediate alingment. Table 3 summarizes the features of the degraded redesigned documentation and compares them to those of the control and redesigned documentation. The design of Study 3 is an experiment similar to Study 2 except that participants are randomly assigned to receive either the degraded redesigned documentation or the redesigned documentation.

Method

*Participants.* Participants were 108 undergraduate students taking a 10-week introductory statistics course at the same university as Study 2. Nine participants were

**Table 3.** Summary of the Three Versions of Documentation.

Documentation Features	Control Documentation (Study 1 and 2)	Degraded Redesigned Documentation (Study 3)	Redesigned Documentation (Study 1, 2, 3)
Example section is forefronted		✓	✓
Examples are ordered by complexity		✓	✓
Example output available		✓	✓
Example output alongside the code (to promote comparison and alignment)			✓

removed from the sample due to incomplete responses or failure to complete the survey in one sitting (i.e., spending more than 10 hours on the entire intervention). Thus, the final sample consisted of 99 participants. The sample demographics reflected the ethnic diversity of the university population: 31.3% White, 45.5% Asian, 8.1% African American, and 15.2% Hispanic or Latino. Due to the COVID-19 pandemic, the course that participants were recruited from was taught in a hybrid format (the first 6 weeks were taught online only and the remaining weeks were taught in person with an online option to Zoom into class). In all cases, the instruction was synchronous, with an instructor there at the same time as the students. Just like in Study 2, students volunteered to participate for extra credit and provided online consent. The study was approved by the institution's IRB.

**Design & Procedure.** Study 3 followed a similar between-subjects experimental design and procedure as Study 2 where participants were randomly assigned to receive either the degraded redesigned documentation or the redesigned documentation. Students were emailed the link to participate in the study by their instructor near the end of the course.

**Materials and Measures.** The materials and measures were the same as in Study 2, except instead of the control documentation, control participants in this study received the degraded version of the redesigned documentation, which required them to click on links to view output in pop-up windows, as explained above (Figure 4).

## Results

There were no significant differences between groups in their composite score of class confidence pre-test before the start of the intervention, indicating successful randomization ( $t(97) = 1.02$ ,  $\eta^2 = .01$ ,  $p = .310$ ).

Similar to Study 2, the documentation had a significant impact on participants' performance in the learning task, as shown in Figure 5. An independent samples t-test found that the redesigned group ( $M = 5.32$ ,  $SD = 2.14$ ) performed significantly better than the degraded redesigned group ( $M = 4.31$ ,  $SD = 1.81$ ),  $t(97) = 2.56$ ,  $\eta^2 = .06$ ,  $p = .012$ . The effect of the documentation remained significant after controlling for students' confidence in the class before the intervention ( $t(96) = 2.38$ ,  $\eta_p^2 = .07$ ,  $p = .019$ ) and confidence also significantly predicted performance ( $t(96) = 2.12$ ,  $\eta_p^2 = .04$ ,  $p = .036$ ).

For questions that involved plots created using functions not mentioned by the documentation, controlling for students' class confidence before the intervention with an ANCOVA, we found a significant effect for condition ( $t(96) = 2.20$ ,  $\eta_p^2 = .06$ ,  $p = .030$ ), but not for class confidence ( $t(96) = 1.95$ ,  $\eta_p^2 = .04$ ,  $p = .054$ ).

## geom\_boxplot: A box and whiskers plot (in the style of Tukey)

### Description

The boxplot compactly displays the distribution of a continuous variable. It visualizes five summary statistics (the median, two hinges and two whiskers), and all "outlying" points individually.

### Examples

```
# This code plots mpg on the y axis and 1 on the x axis. Whatever
comes first inside aes() gets plotted on the x-axis, even constants.
ggplot(mtcars, aes(1, mpg)) +
  geom_boxplot()
```

[Click on this to see the graph output of the code above](#)

```
# This makes side-by-side box plots with factor(cyl) on the x-axis
ggplot(mtcars, aes(factor(cyl), mpg)) +
  geom_boxplot()
```

[Click on this to see the graph output of the code above](#)

```
# Without factor() around cyl, R will give you a warning message
ggplot(mtcars, aes(cyl, mpg)) +
  geom_boxplot()
```

[Click on this to see the graph output of the code above](#)

```
# This code changes which variable appears on the x- and y-axes by
changing what appears first in aes()
ggplot(mtcars, aes(mpg, factor(cyl))) +
  geom_boxplot()
```

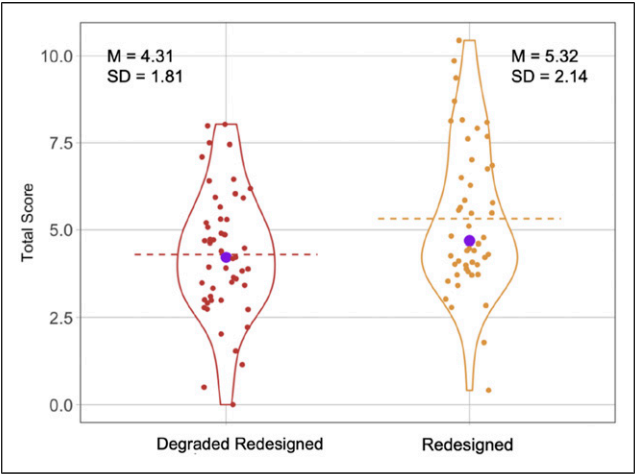
[Click on this to see the graph output of the code above](#)

```
# This adds the argument color to geom_boxplot() to change the outline
color to blue
ggplot(mtcars, aes(factor(cyl), mpg)) +
  geom_boxplot(color = "blue")
```

[Click on this to see the graph output of the code above](#)

**Figure 4.** A screenshot of the modified control documentation.

Similar to Study 2, the two groups of participants did not differ significantly on the three attitudinal measures at the end of the study (Table 4): perceived difficulty ( $t(96) = -1.30$ ,  $\eta^2 = .02$ , 95% CI = [.00, .10],  $p = .196$ ), confidence in their answers ( $t(88) = .58$ ,  $\eta^2 = .00$ , 95% CI = [.00, .07],  $p = .561$ ), or perceived helpfulness of the documentation ( $t(93) = 0.47$ ,  $\eta^2 = .00$ , 95% CI = [.00, .06],  $p = .642$ ).



**Figure 5.** Violin plot of participants’ scores on the coding questions by condition (study 3). The dashed line represents the mean and the purple dot the median of each group.

**Table 4.** Three Attitudinal Measures by Condition in Study 3.

Condition	Difficulty		Confidence		Helpfulness	
	M	SD	M	SD	M	SD
Degraded redesigned	5.86	1.33	2.43	1.31	4.31	4.88
Redesigned	5.83	1.18	2.59	1.23	4.48	1.82

Discussion

Study 3 provides further evidence that a redesigned documentation that scaffolded cognitive processes such as comparison and alignment was more effective than a very similar documentation (degraded redesigned) that did not provide that affordance. The results suggest that the side-by-side presentation of code and output that supports comparison and alignment promotes learning, as predicted by the literature on representational mapping (e.g., Zhang et al., 2024). This finding highlights the importance of integrating code and output visually to reduce cognitive load and enhance learning.

Interestingly, while the redesigned documentation improved performance, it did not significantly affect participants’ attitudinal measures, consistent with the findings from Study 2. This suggests that while effective documentation design can enhance task performance, additional interventions may be needed to influence learners’ perceptions of difficulty, confidence, or helpfulness.

## General Discussion

We proposed that novice programmers differ from expert programmers in that they lack key schematic knowledge related to documentation and programming in general. In three studies, we designed and tested, both qualitatively and quantitatively, the effectiveness of redesigning R documentation to make up for the lack of schemas and scaffold learning. Studies 1 and 2 showed that the redesigned documentation was more effective than a version of the documentation that is very similar to the business-as-usual version in the R community. Study 3 showed that the effectiveness of the redesigned documentation was not merely due to the examples themselves but also to their side-by-side presentation with output, which supported cognitive processes such as mapping and alignment. In summary, novices were more successful at navigating (Study 1) and effectively using the redesigned documentation to make graphs (Studies 2 and 3).

The design changes were hypothesized based on schemas that novice programmers lack (and experts have): a schematic understanding of documentation, causal connections between code and output, and syntactic understanding of the coding language. Here, we will summarize and discuss the key takeaways based on the three studies.

First, our findings showed that novice programmers have not acquired a schema about how R documentation works, which makes them unable to anticipate the structure of the documentation. In Study 1, they read through the documentation in order, without navigating right to the most helpful sections. Examples in the documentation are helpful for novice learners if they can get to the examples. However, the way the documentation is currently structured results in novice programmers reading through unhelpful sections first, and sometimes they give up before they get to the examples.

As novice programmers making the effort to read the long sections of argument and description before the examples, they might be cognitively overloaded, using much of their working memory capacity to try and make sense of content that is largely not helpful to them, to the degree that when they eventually get to the examples, they do not have enough bandwidth to effectively process the examples.

Restructuring the documentation based on how novices will naturally go through it – forefronting the examples – avoids this potential for overload. Such restructuring ensures that novices can get to the most useful part. Experts, because they actually *navigate* documentation, can use the examples if that's what they are looking for, or skip to the part that they need for the task at hand.

Study 1 showed that the redesigned documentation helped novice programmers identify the most relevant examples. Study 2 showed, with a larger sample, that the three features of the Redesigned documentation, taken together, led to superior performance on the learning task. Study 3 in addition showed that just having the output available is not enough for learning. It is most beneficial if the example output is presented alongside the code to facilitate mapping between the two. Perhaps being able to simultaneously and directly compare the code-to-code and



code-to-output leads to a more schema-like understanding of the relationship between code and output.

In situations where learners have to integrate two sources of information before they can comprehend it (e.g., the code and its output), this simultaneous presentation of code and output organizes the documentation to avoid splits of attention (Sweller et al., 2019) and facilitating mapping. Mentally integrating the code and output may create a heavy cognitive load for novice programmers (Mow, 2008) when they need to reserve all the cognitive bandwidth they have for the actual learning, i.e., extracting schemas and patterns between multiple examples. The side-by-side presentation of code and output alleviates the cognitive load associated with holding both code and output *in mind*. Instead, the code and output are perceptually available so that cognitive resources can be used to extract a more schematic understanding of how code causes particular output.

Lastly, not only is the code-to-output comparison important, but it is also critical that novice programmers can easily compare code-to-code. We hypothesized that novice programmers lack a structural understanding of the syntax of the code. Reordering the examples in an incremental fashion might work together with the alignable output to foster both code-to-code comparisons and output-to-output comparisons. Although in Study 1 we did not see a distinct difference between students' ability to modify the code, we did see that students are much more likely and faster to identify the most helpful example. In all three studies, without any documentation for `geom_violin`, students in the redesigned group were more likely to transfer their understanding of `geom_boxplot` to `geom_violin` to create a novel plot. This is evidence that novices can learn isomorphic elements from the documentation and that it makes sense to structure the documentation in a way that facilitates isomorphic comparisons. This finding is consistent with past findings demonstrating the importance of scaffolding deliberate comparisons between representations (e.g., Zhang et al., 2024). Providing learners with opportunities for comparing and contrasting can foster relational knowledge (Gentner et al., 2003; Gentner & Maravilla, 2018; Gray & Holyoak, 2021; Holyoak, 2012), increasing conceptual understanding of the syntax of the code.

### ***Connecting to Code Learning “in the Wild”***

Learning to program can occur in formal settings but also occurs “in the wild” (i.e., outside the classroom). Students and even professionals frequently learn from searching the web for resources such as documentation and Q&A forums (such as Stack Exchange). As generative AI (such as ChatGPT by OpenAI) becomes more broadly available, novices may turn to chatbots to both generate code and explain it. The scholarship around learning to code must include studying how learning occurs in these “wild” settings. The three studies reported here employed a unique combination of qualitative and quantitative methods to explore how novice programmers learn from non-instructional materials they are likely to find in the wild. The qualitative interviews

provided important insights into how novice programmers naturally approach and use documentation when they learn novel functions.

Instructors and curriculum designers should be aware of the challenges students can face when interacting with documentation and other non-instructional materials. One strategy is to help the learners adapt by preparing them for future informal learning, such as teaching them explicitly about how to best search on the internet or use other tools. However, this approach could be difficult to implement if we do not truly understand how our students will struggle in the wild (e.g., what specific barriers they will encounter when they navigate programming documentation they found on the internet). Another adaptive strategy, the one we've taken here, is to consider how the wild environment can be redesigned to anticipate the needs of a broader array of users. Design tweaks to documentation can make it easier for a larger variety of users to navigate and ultimately learn from these non-instructional materials.

### *The Evolution of Documentation*

Our goal is not to criticize current documentation but to show how small tweaks can change its effectiveness. By focusing on the experiences of novices, documentation can be more accommodating for novice learners, without significantly deteriorating the experience of experts. Addressing challenges faced by novice programmers may also broaden the community of people pursuing programming in informal learning settings.

In fact, efforts to document functions in the R and other programming language communities have converged on similar design changes (forefronting examples, providing example output, ordering examples by complexity), presumably because the developers were trying to “educate” new users about these functions. For example, tidyverse has “reference” pages for its functions (e.g., [https://ggplot2.tidyverse.org/reference/geom\\_boxplot.html](https://ggplot2.tidyverse.org/reference/geom_boxplot.html)) which provide example output and order the examples by complexity. The R community has created other resources (e.g., cheatsheets) to complement documentation, which also provides output or schematic drawings of data frames alongside example code. These design changes may have been made intuitively by the tidyverse creators, but our studies have begun to provide empirical evidence that these changes are indeed improvements.

There are also broader systemic efforts to revolutionize the design of documentation. Systematic frameworks for writing documentation, such as Diátaxis (see <https://diataxis.fr/>), emphasize the users' needs when developing the documentation. Consistent with our analysis of expert schemas, Diátaxis highlights the problem of structure: developers should consider how to structure the documentation based on user needs. Diátaxis assumes that users will have different needs and provides “a map of distinct documentation types” to help developers anticipate different user needs.

Documentation is an ever-evolving community product. Our project in chorus with the approaches of others in the community encourages us to think outside the box of the initial purpose of documentation. Even though documentation was intended for communicating with experts, its availability on the internet has attracted other users.

An iterative process of design thinking based on user needs can help make documentation more effective for both novices and experts.

### *Limitations and Future Directions*

The three studies reported here serve as an initial exploration of the proposed theoretical account: that novices are missing key schemas that hinder their navigation and learning from the documentation. Although these studies provide some evidence that the design changes based on these missing schemas help novices, we did not directly examine whether novices gained a more schematic knowledge of documentation or the code itself. Future studies should take a closer and more longitudinal look into whether navigating documentation and attempting to learn from it helps students develop more accurate schemas. In addition, although the current investigation focused on novice learners, future studies might be interested in exploring whether other groups of users, such as experts, might also benefit from these redesigns, and whether these effects generalize to different programming tasks.

There are some other limitations of this project that we want the readers to bear in mind. First, students did not run any code during this project but running and testing the codes is an indispensable part of programming. Future studies should examine how the use of documentation might be different when students can toggle between testing out code and referring to the documentation. Second, although our results suggest that the redesigned documentation is an improvement, the redesigned version is not perfect nor is it a substitute for formal learning (e.g., tutorials, books, courses). Documentation alone may not be sufficient for helping students gain a functional understanding of coding, but our goal is to examine the part it might play in informal learning. There are also individual differences in how students use documentation – some students immediately saw the benefit of examples while others did not. Future studies should consider how students navigate formal and informal learning situations and also consider the role of individual differences in these two different contexts.

### **Conclusion**

What distinguishes novices and experts? Experts can take advantage of sub-optimally designed learning opportunities, but this is hard for novices. Programming documentation was not intended as a learning tool for novices. Yet, because it is widely available and often among the top internet search results, novices will attempt to learn from it. By studying how novices use documentation, we were able to propose, implement, and test design changes that might benefit novices. These insights can inform the design of future information technologies. We hope these studies serve as a starting point for both researchers and developers to think about how to make new information technologies easier to be learned and adopted in informal learning settings such as learning from documentation.

Most of us will have to engage in informal learning beyond the classroom. This project demonstrates the importance of bridging basic research and theories in psychology to the opportunities for learning in real life. Because technology always outpaces human evolution, we need more research on how psychological processes intertwine with the world around us, the internet, AI, and more. At the same time, what we know about basic cognitive processes can help us shape the world we live in. We can build a better learning environment and simultaneously learn more about ourselves by testing our theories of cognitive mechanisms in situations of authentic, self-guided learning.

### Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

### Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This project has been made possible in part by a grant from the Chan Zuckerberg Foundation.

### ORCID iD

Icy (Yunyi) Zhang  <https://orcid.org/0000-0003-3423-6794>

### Supplemental Material

Supplemental material for this article is available online.

### References

- Bakar, M. A., Mukhtar, M., & Khalid, F. (2019). The development of a visual output approach for programming via the application of cognitive load theory and constructivism. *International Journal of Advanced Computer Science and Applications*, 10(11). <https://doi.org/10.14569/ijacsa.2019.0101142>
- Begel, A., & Ko, A. (2019, March 30). *Learning outside the classroom* (pp. 749–772). <https://doi.org/10.1017/9781108654555.027>. Cambridge University Press.
- Bosse, Y., & Gerosa, M. A. (2017). Difficulties of programming learning from the point of view of students and instructors. *IEEE Latin America Transactions*, 15(11), 2191–2199. <https://doi.org/10.1109/tla.2017.8070426>
- Butcher, K. R. (2006). Learning from text with diagrams: Promoting mental model development and inference generation. *Journal of educational psychology*, 98(1), 182.
- Camp, T., Zweben, S., Walker, E., & Barker, L. (2015). Booming enrollments: Good times? In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, Kansas City, Missouri, USA, 4–7 March, 2015, pp. 80–81.

- Carbonell, K. B., Stalmeijer, R. E., Könings, K. D., Segers, M., & van Merriënboer, J. J. (2014). How experts deal with novel situations: A review of adaptive expertise. *Educational Research Review*, 12, 14–29. <https://doi.org/10.1016/j.edurev.2014.03.001>
- Chandler, P., & Sweller, J. (1996). Cognitive load while learning to use a computer program. *Applied Cognitive Psychology*, 10(2), 151–170. [https://doi.org/10.1002/\(sici\)1099-0720\(199604\)10:2<151::aid-acp380>3.0.co;2-u](https://doi.org/10.1002/(sici)1099-0720(199604)10:2<151::aid-acp380>3.0.co;2-u)
- Chi, M. T. (2011). Theoretical perspectives, methodological approaches, and trends in the study of expertise. In *Expertise in mathematics instruction* (pp. 17–39). Springer.
- Clement, C. A., Kurland, D. M., Mawby, R., & Pea, R. D. (1986). Analogical reasoning and computer programming. *Journal of Educational Computing Research*, 2(4), 473–486. <https://doi.org/10.2190/DFH5-E0PG-1ML4-M34J>
- Cogo, F. R., Xia, X., & Hassan, A. E. (2022). Assessing the alignment between the information needs of developers and the documentation of programming languages: A case study on rust. *ACM Transactions on Software Engineering and Methodology*, 32(2), 1–48. <https://doi.org/10.1145/3546945>
- Crk, I., Kluthe, T., & Stefik, A. (2015). Understanding programming expertise: An empirical study of phasic brain wave changes. *ACM Transactions on Computer-Human Interaction*, 23(1), 1–29. <https://doi.org/10.1145/2829945>
- Dasgupta, S., & Hill, B. M. (2017, April). Learning to code in localized programming languages. In Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale, Cambridge, MA, USA, 20–21 April, 2017, pp. 33–39.
- Denny, P., Luxton-Reilly, A., Tempero, E., & Hendrickx, J. (2011). Understanding the syntax barrier for novices. In Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, Darmstadt, Germany, 27–29 June, 2011, pp. 208–212.
- Eghterafi, W., Tucker, M. C., Zhang, I. Y., & Son, J. Y. (2022). Instructor presence, instructor effectiveness, retention, student success, Community of Inquiry (CoI). *Online Learning*, 26(2). <https://doi.org/10.24059/olj.v26i2.2731>
- Ericsson, K. A., Hoffman, R. R., Kozbelt, A., & Williams, A. M. (Eds.), (2018). *The Cambridge handbook of expertise and expert performance*. Cambridge University Press.
- Fries, L., Son, J. Y., Givvin, K. B., & Stigler, J. W. (2021). Practicing connections: A framework to guide instructional design for developing understanding in complex domains. *Educational Psychology Review*, 33(2), 739–762. <https://doi.org/10.1007/s10648-020-09561-x>
- Garner, S. (2002). *Reducing the cognitive load on novice programmers* (pp. 578–583). Association for the Advancement of Computing in Education (AACE).
- Gentner, D., Loewenstein, J., & Thompson, L. (2003). Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology*, 95(2), 393–408. <https://doi.org/10.1037/0022-0663.95.2.393>
- Gentner, D., & Maravilla, F. (2018). Analogical reasoning. In L. J. Ball & V. A. Thompson (Eds.), *International handbook of thinking and reasoning* (pp. 186–203). Routledge.
- Gentner, D., Shao, R., Simms, N., & Hespos, S. (2021). Learning same and different relations: Cross-species comparisons. *Current Opinion in Behavioral Sciences*, 37, 84–89. <https://doi.org/10.1016/j.cobeha.2020.11.013>

- Gomes, A., & Mendes, A. J. (2007, September). Learning to program-difficulties and solutions. In *International Conference on Engineering Education-ICEE* (Vol. 7). CEE.
- Gray, M. E., & Holyoak, K. J. (2021). Teaching by analogy: From theory to practice. *The Official Journal of the International Mind, Brain, and Education Society*, 15(3), 250–263. <https://doi.org/10.1111/mbe.12288>
- Gross, P., & Powers, K. (2005). Evaluating assessments of novice programming environments. In *Proceedings of the First International Workshop on Computing Education Research*, Seattle, WA, USA, 1–2 October, 2005, pp. 99–110.
- Holyoak, K. J. (2012). Analogy and relational reasoning. In K. J. Holyoak & R. G. Morrison (Eds.), *The Oxford handbook of thinking and reasoning, Oxford library of psychology* (pp. 234–259). Oxford Academic. <https://doi.org/10.1093/oxfordhb/9780199734689.013.0013>
- Holyoak, K. J., & Cheng, P. W. (2011). Causal learning and inference as a rational process: The new synthesis. *Annual Review of Psychology*, 62(1), 135–163. <https://doi.org/10.1146/annurev.psych.121208.131634>
- Ichinco, M., & Kelleher, C. (2015). Exploring novice programmer example use. In 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Atlanta, GA, USA, 18–22 October 2015, pp. 63–71. IEEE.
- Iwamoto, T., Matsumoto, S., Yamagishi, S., & Kashima, T. (2020). Pre-and post-survey of the achievement result of novice programming learners-on the basis of the scores of puzzle-like programming game and exams after learning the basic of programming. In *Transactions on engineering technologies: International MultiConference of engineers and computer scientists 2018* (pp. 130–142). Springer.
- Jadud, M. C. (2005). A first look at novice compilation behaviour using BlueJ. *Computer Science Education*, 15(1), 25–40. <https://doi.org/10.1080/08993400500056530>
- Jeffries, R. (1981). The processes involved in designing software. In *Cognitive skills and their acquisition* (pp. 255–283). Psychology Press.
- Jeong, S. Y., Xie, Y., Beaton, J., Myers, B. A., Stylos, J., Ehret, R., Karstens, J., Efeoglu, A., & Busse, D. K. (2009). Improving documentation for eSOA APIs through user studies. In *End-User Development, 2nd International Symposium, IS-EUD 2009, Siegen, Germany, 2–4 March, 2009*, 86–105. [https://doi.org/10.1007/978-3-642-00427-8\\_6](https://doi.org/10.1007/978-3-642-00427-8_6)
- Kalyuga, S., Chandler, P., & Sweller, J. (1998). Levels of expertise and instructional design. *Human Factors*, 40(1), 1–17. <https://doi.org/10.1518/001872098779480587>
- Kaplan, D., & Pruim, R. (2022). ggformula: Formula interface to the grammar of graphics. R package version 0.10.2. <https://github.com/ProjectMOSAIC/ggformula>
- Kashima, T. (2019, October). Pre-and post-survey of the achievement result of novice programming learners-on the basis of the scores of puzzle-like programming game and exams after learning the basic of programming. In *Transactions on engineering technologies: International MultiConference of engineers and computer scientists 2018* (p. 130). Springer Nature.
- Kotovskiy, L., & Gentner, D. (1996). Comparison and categorization in the development of relational similarity. *Child Development*, 67(6), 2797–2822. <https://doi.org/10.1111/j.1467-8624.1996.tb01889.x>

- Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4), 119–150. <https://doi.org/10.1145/1041624.1041673>
- Luxton-Reilly, A. (2016). Learning to program is easy. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, Arequipa, Peru, 11–13 July, 2016, pp. 284–289.
- Martin, W., Silander, M., & Rutter, S. (2019). Digital games as sources for science analogies: Learning about energy through play. *Computers & Education*, 130, 1–12. <https://doi.org/10.1016/j.compedu.2018.11.002>
- McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2003). The impact of pair programming on student performance, perception and persistence. In 25th International Conference on Software Engineering, Portland, OR, USA, 03–10 May 2003. <https://doi.org/10.1109/icse.2003.1201243>
- Mehmood, E., Abid, A., Farooq, M. S., & Nawaz, N. A. (2020). Curriculum, teaching and learning, and assessments for introductory programming course. *IEEE Access*, 8, 125961–125981. <https://doi.org/10.1109/access.2020.3008321>
- Mow, I. T. C. (2008). Issues and difficulties in teaching novice computer programming. In *Innovative techniques in instruction technology, E-learning, E-assessment, and education* (pp. 199–204). Springer. [https://doi.org/10.1007/978-1-4020-8739-4\\_36](https://doi.org/10.1007/978-1-4020-8739-4_36)
- Peng, J., Wang, M., Sampson, D., & van Merriënboer, J. J. G. (2019). Using a visualisation-based and progressive learning environment as a cognitive tool for learning computer programming. *Australasian Journal of Educational Technology*, 35(2). <https://doi.org/10.14742/ajet.4676>
- Perera, P., Tennakoon, G., Ahangama, S., Panditharathna, R., & Chathuranga, B. (2021). A systematic mapping of introductory programming languages for novice learners. *IEEE Access*, 9, 88121–88136. <https://doi.org/10.1109/access.2021.3089560>
- Perkins, D. N., Schwartz, S., & Simmons, R. (2013). Instructional strategies for the problems of novice programmers. In *Teaching and learning computer programming* (pp. 153–178). Routledge.
- Piteira, M., & Costa, C. (2013). Learning computer programming: Study of difficulties in learning programming. In Proceedings of the 2013 International Conference on Information Systems and Design of Communication, Lisbon, Portugal, 11 July 2013, pp. 75–80. <https://doi.org/10.1145/2503859.2503871>
- Qualtrics. (2022). <https://www.qualtrics.com>
- R Core Team. (2021). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. <https://www.R-project.org/>
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2), 137–172. <https://doi.org/10.1076/csed.13.2.137.14200>
- Robins, A. V. (2019). 12 Novice programmers and introductory programming. In *The Cambridge handbook of computing education research* (p. 327). Cambridge University Press.

- Sarkar, S. P., Sarker, B., & Hossain, S. A. (2016). Cross platform interactive programming learning environment for kids with edutainment and gamification. In 2016 19th International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 18–20 December 2016, pp. 218–222. IEEE. <https://doi.org/10.1109/iccitechn.2016.7860198>
- Scheiter, K., & Eitel, A. (2015). Signals foster multimedia learning by supporting integration of highlighted text and diagram elements. *Learning and Instruction*, 36, 11–26. <https://doi.org/10.1016/j.learninstruc.2014.11.002>
- Schmidt-Weigand, F., Kohnert, A., & Glowalla, U. (2010). Explaining the modality and contiguity effects: New insights from investigating students' viewing behaviour. *Applied Cognitive Psychology: The Official Journal of the Society for Applied Research in Memory and Cognition*, 24(2), 226–237.
- Shilane, D., Di Crecchio, N., & Lorenzetti, N. L. (2024). Some pedagogical elements of computer programming for data science: A comparison of three approaches to teaching the R language. *Teaching Statistics*, 46(1), 24–37. <https://doi.org/10.1111/test.12361>
- Soosai Raj, A. G., Ketsuriyonk, K., Patel, J. M., & Halverson, R. (2018). Does native language play a role in learning a programming language? In Proceedings of the 49th ACM Technical Symposium on Computer Science Education, Baltimore, MD, USA, 21–24 February 2018, (pp. 417–422).
- Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computer Education*, 13(2), 1–31. <https://doi.org/10.1145/2483710.2483713>
- Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2), 257–285. [https://doi.org/10.1016/0364-0213\(88\)90023-7](https://doi.org/10.1016/0364-0213(88)90023-7)
- Sweller, J. (2010). Cognitive load theory: Recent theoretical advances.
- Sweller, J., van Merriënboer, J. J., & Paas, F. (2019). Cognitive architecture and instructional design: 20 years later. *Educational Psychology Review*, 31(2), 261–292. <https://doi.org/10.1007/s10648-019-09465-5>
- Tan, P. H., Ting, C. Y., & Ling, S. W. (2009). Learning difficulties in programming courses: Undergraduates' perspective and perception. In 2009 International Conference on Computer Technology and Development, Kota Kinabalu, Malaysia, 13–15 November 2009, Vol. 1, pp. 42–46. IEEE. <https://doi.org/10.1109/icctd.2009.188>
- Thompson, C. A., & Opfer, J. E. (2010). How 15 hundred is like 15 cherries: Effect of progressive alignment on representational changes in numerical cognition. *Child Development*, 81(6), 1768–1786. <https://doi.org/10.1111/j.1467-8624.2010.01509.x>
- Tucker, M. C., Wang, X. W., Son, J. Y., & Stigler, J. W. (2024). Prediction versus production for teaching computer programming. *Learning and Instruction*, 91(9), Article 101871. <https://doi.org/10.1016/j.learninstruc.2023.101871>
- van der Meij, J., & de Jong, T. (2006). Supporting students' learning with multiple representations in a dynamic simulation-based learning environment. *Learning and Instruction*, 16(3), 199–212. <https://doi.org/10.1016/j.learninstruc.2006.03.007>
- Wickham, H. (2016). Data analysis. In *ggplot2* (pp. 189–201). Springer.
- Wiedenbeck, S. (1985). Novice/expert differences in programming skills. *International Journal of Man-Machine Studies*, 23(4), 383–390. [https://doi.org/10.1016/s0020-7373\(85\)80041-9](https://doi.org/10.1016/s0020-7373(85)80041-9)



- Wiedenbeck, S., Fix, V., & Scholtz, J. (1993). Characteristics of the mental representations of novice and expert programmers: An empirical study. *International Journal of Man-Machine Studies*, 39(5), 793–812. <https://doi.org/10.1006/imms.1993.1084>
- Zhang, I. Y., Gray, M. E., Cheng, A. X., Son, J. Y., & Stigler, J. W. (2024). Representational-mapping strategies improve learning from an online statistics textbook. *Journal of Experimental Psychology: Applied*, 30(2), 293–317. <https://doi.org/10.1037/xap0000474>
- Zweben, S., & Bizot, B. (2021). 2020 CRA Taulbee survey. *Computing Research News*, 33(5), 2–68.

## Author Biographies

**Icy(Yunyi) Zhang** is an assistant professor in Learning Sciences in the Department of Educational Psychology at the University of Wisconsin-Madison. Icy is interested in using psychological theories to inform real-world educational practices. She is particularly interested in empowering disadvantaged learners through innovative pedagogies and technologies. Her areas of interest include embodied learning, representational mapping and statistics and data science education.

**Yunqi (Ruby) Jia** holds a Bachelor of Arts in Psychology and a Master of Education in Student Affairs from UCLA. She specializes in advising, assessment, and creating inclusive educational environments. Committed to empowering diverse student communities, Ruby focuses on fostering resilience and persistence in higher education through a student-centered approach.

**Xiaoxuan (Alicia) Cheng** holds a Bachelor's degree in Cognitive Science from UCLA and is currently pursuing a Master's degree in Human Factors and Human-Computer Interaction at Rice University. Her research and practice focus on using human-centered design processes to ensure that systems and products are developed with users in mind.

**Ji Y. Son** is Professor of Psychology and Director of the Learning Lab at California State University, Los Angeles. Along with James Stigler (UCLA), she co-authored the interactive textbook “Statistics and Data Science: A Modeling Approach” (available at CourseKata.org). She uses and develops new technology to innovate teaching, learning, and educational research to improve student outcomes particularly for marginalized students. In the lab and classroom, she focuses on basic cognitive and perceptual processes that can foster rich and transferable learning. The central idea behind Ji's work is that learning changes the way we see the world.

**James W. Stigler** is a psychologist, researcher, entrepreneur, and author. He is Distinguished Research Professor of Psychology at UCLA, was director of the TIMSS video studies, and the founder of three education technology companies: LessonLab (acquired by Pearson Education in 2003); Zaption (acquired by Workday in 2016); and co-founder in 2017 of CourseKata.org, a nonprofit that develops and improves research-based interactive online textbooks for colleges and high schools. Dr. Stigler's current work focuses on the teaching and learning of mathematics and statistics, and on developing and testing new approaches to education research and development.