Title
Prediction versus Production for Teaching Computer Programming

Author
Tucker, Mary Conyers

Publication Date
2022

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Prediction versus Production for Teaching Computer Programming

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in Psychology

by

Mary Conyers Tucker

2022

ABSTRACT OF THE DISSERTATION

Prediction versus Production for Teaching Computer Programming

by

Mary Conyers Tucker

Doctor of Philosophy in Psychology

University of California, Los Angeles, 2022

Professor James W. Stigler, Chair

Learning to program is increasingly important. Yet, it is becoming clear that most students struggle when learning to program (McCracken et al., 2001). This is leading to a divide where some people can program but many others can't. Prior research has traced poor student outcomes to their early experiences learning programming. Still, little is known about how different programming tasks might impact the processes involved in learning programming. In this dissertation, I extend prior research on students' experiences when learning computer programming and build on this research by testing the causal influences of different learning tasks on students' emotions, motivation, and learning. In a randomized experiment, I manipulated the instructional tasks used to introduce programming and investigated students' emotional trajectories, motivation, and learning outcomes. Participants randomly assigned to predict the outcome of pre-provided code showed more positive emotional trajectories, increased motivation, and greater learning compared to students randomly assigned to modify or produce their own code.

The dissertation of Mary Conyers Tucker is approved.

Ji Son

Jennifer Ashley Silvers

Keith Holyoak

James W. Stigler, Committee Chair

University of California, Los Angeles

2022

This dissertation is dedicated to the memory of Olivia McManaway, Herman McManaway, Barbara Tucker, and Donald Tucker Sr., my loving grandparents.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

Many helped me along this journey. First, I would like to thank the UCLA undergraduate students who inspired this work and who participated in this research. Second, I am ever grateful for my graduate advisors – Jim Stigler and Ji Son – both brilliant researchers and dedicated teachers and mentors. Their kindness, support, and guidance enabled me to cross the finish line on this dissertation and prepared me for the next stage in my career. I could not have asked for better advisors. I would also like to thank Karen Givvin, Stacy Shaw, Laura Fries, Claudia Sutter, Adam Blake, and the entire Teaching and Learning Lab for their support, Xinran Wang, whose work as a Research Assistant helped get this project off the ground, as well as Jennifer Silvers and Keith Holyoak for their valuable feedback on this dissertation. Finally, I'd like to thank my partner, Zach, my parents, and my brother, Don, for their unwavering support and encouragement. I could not have done this without them.

Mary C. Tucker

EDUCATION

2017-*present* **University of California, Los Angeles**
Ph.D., Developmental Psychology (expected 2022)
M.A., Psychology

2006-2010 **Duke University**
B.A., Psychology with Honors

RESEARCH POSITIONS

2017-2022 **Teaching and Learning Lab**, University of California, Los Angeles
Research Fellow

2015-2016 **Stanford Psychophysiology Lab**, Stanford University
Research Assistant

PROFESSIONAL EXPERIENCE

2022-*present* **Barracuda Networks**, Campbell, CA
Sr User Researcher

2014-2015 **Virtual Learning Technologies (Sokikom)**, Mountain View, CA
Director of Teacher Happiness

2012-2014 **KIPP: TEAM Academy**, Newark, NJ
Math Learning Specialist

2010-2012 **Teach for America / Harrison School District II**, Colorado Springs, CO
Special Education Teacher

HONORS & AWARDS

2017-2019 Graduate Dean's Scholar Award, University of California, Los Angeles
2017 Psychology Departmental Fellowship, University of California, Los Angeles
2011-2012 Segal Education Award, Americorps
2010 Graduation with Distinction, Duke University
2009 Undergraduate Research Support Grant, Duke University
2009 Vertical Integration Program Summer Research Fellowship, Duke University

PUBLICATIONS

Peer-reviewed journal articles

Zhang, Y., Tucker, M.C., & Stigler, J.W. (2022). Watching a hands-on activity improves students' understanding of randomness. *Computers & Education, 186*(3).

Tucker, M.C., Shaw, S., Son, J.Y. & Stigler, J.W. (2022). Teaching statistics and data analysis with R. *Journal of Statistics and Data Science Education*.

Eghterafi, W., Tucker, MC., Zhang, Y., & Son, J.Y. (2022). Effect of feedback with video-based peer modeling on learning and self-efficacy. *Online Learning*.

Sutter, C., Hulleman, C., Givvin, K.B., & Tucker, M.C. (2021). Utility value trajectories and their relationship with behavioral engagement and performance in introductory statistics. *Learning and Individual Differences*.

Stigler, J.W., Son, J.Y., Givvin, K.B., Blake, A.B., Fries, L., Shaw, S. & Tucker, M.C. (2020). The better book approach for education research and development. *Teachers College Record, 122*(9).

Conference papers

Sutter, C.C., Tucker, M.C., Givvin, K.B., & Hulleman, C. (2022). How much students value an introductory statistics course, how value levels change across the term, and how they predict learning. *IASE 2021 Satellite Conference: Statistics Education in the Era of Data Science*.

Zhang, Y., Tucker, M.C., & Stigler, J.W.(2022). Watching hands shuffle data improves subsequent understanding of R-based simulations of randomness. *IASE 2021 Satellite Conference: Statistics Education in the Era of Data Science*.

Tucker, M.C., Shaw, S., Son, .Y., & Stigler, J.W. (2021). Integrating R in a college statistics course improves student attitudes toward programming. *Annual Meeting of the American Educational Research Association.*

Prediction versus Production for Teaching Computer Programming

Computer programming is increasingly important, but programming is often viewed as challenging by undergraduate students. These challenges are particularly pronounced for students with no prior programming experiences and those belonging to groups traditionally underrepresented in STEM fields (Salguero et al., 2021; Luxton-Reilly, 2016; Robins et al., 2003; Watson & Li, 2014). Even successful graduates of introductory courses struggle to learn programming in the future. Many students learn to code by following steps, like a cookbook; but fewer really develop transferable and flexible knowledge of computer programming (Perkins & Martin, 1986).

Given the growing demand for people who can do computer programming, and the relatively few students who develop transferable programming knowledge, issues of students' early experiences and outcomes learning programming are important to address. Other researchers have identified the issue of students' challenges learning programming and attempted to address them using specially designed programming software and interventions (i.e., Simon & Hanks, 2008). In this dissertation, I take a different approach; changing the structure of the tasks in which students are first introduced to programming. Extensive research has shown that learning environments can have a profound impact on students' cognition, emotions, and motivated behavior. Yet, few studies have attempted to manipulate the structure of programming tasks as a way to cultivate more flexible knowledge and adaptive approaches to learning programming.

Psychological Research on Code Writing and Learning Computer Programming

Computer programming is a complex endeavor with cognitive and affective challenges. To develop flexible programming skills, students need to learn programming syntax (Altadmri & Brown, 2015), understand programming concepts (Bayman & Mayer, 1983; Cañas et al., 1994;

Ma, 2008; Sirkiä & Sorva, 2012), and coordinate and apply this knowledge to solve novel problems (Clancy & Linn, 1999; Davies, 1993 a, b; Lister et al., 2004; Lister et al., 2006; Perkins and Martin, 1986; Sajaniemi & Navarro-Prieto, 2005; Whalley et al., 2006) (see Quian & Lehman, 2017 for a review). Students also need to regulate emotions that arise during learning (Bosch et al., 2013; D'Mello, 2011; D'Mello & Graesser, 2011), maintain motivation and engagement, and persist in the face of failure (Renumol et. al, 2010).

Carefully designed programming instruction can help students overcome these challenges. But some programming instruction may increase students' difficulties. For example, most undergraduate courses include code writing activities in which students follow task instructions to modify or write their own code. While code writing seems like a logical starting point for teaching programming, decades of research documenting students' poor programming outcomes bring the effectiveness of this strategy into question. Below, I discuss the effect of traditional code writing tasks on student outcomes and the processes through which they may exert their effects.

Effect of Code Writing on Emotions

One way code writing tasks might prevent flexible knowledge acquisition is by eliciting negative emotions during learning. Emotions arise during all types of learning but are especially prevalent during complex learning in domains like computer programming, where students encounter novel and difficult problems and struggle to comprehend and make sense of abstract concepts (D'Mello & Graesser, 2011). In most code writing assignments, students are allowed to progress only after they submit code that runs correctly. However, research has shown that students experience high rates of failure on code writing assignments (McCracken et al., 2001), which elicits frustration and other negative emotions. If students continue to try new solutions and fail, they may experience repeated failure and negative feedback. Repeated experiences of

failure can lead to frustration, which leads to disengagement and boredom, a process that D'Mello and Graesser refer to as the "vicious cycles" of negative emotions (D'Mello & Graesser, 2011, p. 15).

This hypothesis is supported by studies of affect dynamics, which show that writing and submitting code is a central event that precedes and follows students' emotions when learning computer programming (Bosch & D'Mello, 2008; Bosch et al., 2013). In one relevant study, Bosch et al. (2013) systematically linked students' emotions to three behaviors during learning: constructing (writing code), running (clicking "Run" or "Submit" to execute the code), or idling (not otherwise interacting with the interface). Students experienced higher rates of frustration when they were running code. Similarly, whereas writing code that runs without errors preceded positive emotions, writing code and getting an error preceded negative affective states (e.g., confusion and frustration).

Beyond reducing students' enjoyment of learning computer programming, negative emotions can work through cognitive and motivational processes to impact knowledge acquisition and future behavior. According to cognitive resource allocation models (Ellis & Ashbrook, 1989), emotions that are not related to the task at hand (i.e., being frustrated about getting the answer incorrect) can take up cognitive resources and distract from learning while positive emotions related to the learning task can focus attention on the activity. Emotional reactions can also activate processes which are implicated in belief formation and learning. For instance, the affect-as-information model holds that emotional reactions provide information about the objects to which the individual is reacting and thus influence beliefs about that object (Schwartz & Clore, 1983). Finally, emotions have been shown to influence the strategies and approaches students use during learning. While positive affective states are thought to facilitate

more flexible and creative approaches to solving problems, negative affective states are thought to lead to more rigid and analytical thinking (Isen, 2000).

Effect of Code Writing on Motivation

Another way code writing might impact flexible learning is by influencing students' motivation. According to expectancy, value, cost models of motivation (Barron & Hulleman, 2015), students' perceptions of how costly an activity is in terms of the time it takes, how stressful it is, and how much it takes away from participation in other valued activities impacts their motivation and future engagement. If students struggle on early programming tasks, they may perceive learning programming as more costly which can influence their approaches to learning and their motivation to learn programming in the future.

Over time, repeated failures can negatively affect students' self-efficacy (Bandura, 1977) – students' judgments of their personal capability to achieve a desired goal. Self-efficacy regulates functioning through cognitive, motivational, emotional, and decision-making processes. Individuals with high self-efficacy pursue more ambitious goals, exert more effort towards those goals, persist longer in the face of setbacks and failure, and ultimately demonstrate deeper learning. Like emotions, self-efficacy is especially important in complex learning environments, where learners face frequent obstacles and frustrations (Davis et al., 1998). In the context of computer education, computer efficacy beliefs have been shown to influence learners' decisions to use computers (Hill et al., 1987), and students with high computer efficacy learned more in an introductory course than students with low computer efficacy beliefs (Martocchio & Dulebohn, 1994; Martocchio & Judge, 1997; Webster & Martocchio, 1992). Self-efficacy has also been implicated in studies of how people learn computer programming.

Finally, programming tasks might influence students' learning and motivation to learn by eliciting and enhancing maladaptive beliefs about the nature of computer programming. If

students only experience programming as writing and submitting code, they may develop simplistic conceptions of what it means to "do" and what it means to "learn" programming. For instance, Braune and Mühling (2020) found that, to students, the programming experienced during learning was less complex and less representative of what it means to do programming "in the real world". The authors suggest that the simplified design of educational programming environments may limit students' understanding of what programming is and what it means to do programming outside the classroom. In line with expectancy value models of achievement motivation (e.g., Wigfield, 1994), if students hold simplistic beliefs about what it means to do programming, they may come to value learning programming less, and thus be less motivated to learn programming in the future and less likely to persist in the face of failure.

Students may also develop other maladaptive beliefs that impede future learning and engagement. For example, among college students, programming has a reputation of being "difficult", which can reduce students' confidence and even deter students from engaging in programming (Medeiros et al., 2019). As past research has shown, students' conceptions of programming can also affect the strategies they adopt and how they interpret events during learning. For instance, in a sample of 421 Taiwanese students majoring in computer science, Liang et al. (2015) found that students who conceived of programming as "memorization" and "rote learning" showed more surface-level motivation and approaches to learning. Umapathy et al. (2020) replicated these findings in a study with computer science students in the United States. Their findings further indicated that students' who conceived of programming as "memorization" were more extrinsically motivated than those who held more complex beliefs.

Effect of Code Writing on Cognitive Processing

Finally, traditional code writing tasks may influence knowledge acquisition directly by the depth of cognitive processing they afford. Traditional programming instruction often makes

use of code writing tasks in which students are first explicitly taught concepts and then asked to apply that knowledge to modify or write code. The format of this instruction is similar to what Schwartz and colleagues (2004; 2011) have described as a "tell and practice" approach in which students are provided explicit instruction and then asked to practice what they have learned. One problem with this approach is that it enables more surface level cognitive processing and approaches to learning compared to elaborative instructional strategies that direct students' attention to deeper structural components of problems.

The sequence of coding instruction may also influence cognitive processing required for developing transferable knowledge. For example, prior research suggests that code tracing – the ability to read and interpret what code does – is a precursor to relational thinking and problem solving – skills that are central to code writing. (Lister et al., 2004; Philpott, Robbins and Whalley et al., 2006; Lopez et al., 2008; Lister et al., 2009). Code reading is also central to "debugging", a central task of programming. Despite the importance of code comprehension, most introductory programming courses do not directly teach this skill. By introducing code writing before reading and understanding how code works, traditional programming activities may introduce complexity, increase the inherent task difficulty, and impede students' abilities to read code in the future.

Another way in which code writing tasks might prevent flexible understanding is by introducing extraneous cognitive load. Code writing activities include elements of interactivity over and above other learning tasks. A common assumption is that the interactivity will benefit learning because it increases student engagement. Another assumption is that interactivity will facilitate activities like "tinkering" and exploration, which are important for learning computer programming. However, adding interactivity can increase the cognitive burden of learning and inhibit knowledge acquisition and transfer (Sweller, 2010).

In Cognitive Load Theory, working memory is a limited resource, which is impacted by the inherent complexity of the learning material and the way instruction is designed (Sweller, 1994). Intrinsic cognitive load concerns the inherent complexity of the information to be learned, independent of the instructional design. Extraneous cognitive load refers to complexity induced by the instructional materials, irrespective of the inherent difficulty of the content. A key concept of intrinsic and extraneous cognitive load is element interactivity, which refers to the level of dependency of the elements – information or concepts – to be learned. Tasks with low element interactivity can be accomplished independently and thus require less working memory. Tasks with high element interactivity can only be accomplished with knowledge of other elements or information and require more working memory (Sweller, 2010).

Computer programming has high element interactivity because it requires students to understand multiple elements, including programming syntax, programming concepts, and problem-solving approaches. Code writing tasks add even more element interactivity because they require students to understand the programming environment on top of syntax, concepts, and strategy. For example, students need to understand what different keystrokes do within the programming environment, to interpret error messages, how to resolve technical difficulties, and how to submit their code. If the goal of learning programming in the early stages is to understand what code does, adding elements of interactivity through code writing tasks could potentially increase the extraneous cognitive load and impede meaningful learning (Schnotz & Kürschner, 2007).

Taken together, prior research suggests that code writing tasks may impede meaningful learning through cognitive, emotional, and motivational processes. There are a number of ways to potentially interrupt these processes. For example, researchers have attempted to manipulate students' programming mindsets directly through growth mindset interventions. Other

researchers have attempted to reduce the cognitive load of learning computer programming by designing specialized programming languages for novice learners (see Resnick et al., 2009 for an example).Though targeted interventions and programming languages may be a good option in some contexts, they are not always practical for teachers to implement into existing instructional routines. Another possible option is to implement theoretically informed changes to the structure of the programming task so that students acquire deeper knowledge, have more positive emotional experiences, and develop more adaptive beliefs about what it means to learn programming.

Limitations to Prior Work

Our understanding of students' experiences learning computer programming has advanced dramatically over the past ten years. Still, there are several limitations in studies of students' experiences learning programming that are important to address. First, prior research indicates that students exhibit poor learning and transfer and maladaptive beliefs at the end of computer programming courses. However, the mechanisms and conditions under which students acquire programming knowledge and maladaptive beliefs have not been studied. Given the well-established link between emotions and cognition in other academic contexts and the documented link between computer programming interactions, emotions, and performance, emotions may be one plausible mechanism through which early experiences learning programming might influence students' knowledge acquisition and motivation to learn programming.

Second, most of what we know about students' experiences learning programming is based on research conducted with students in introductory programming courses. Yet, students are increasingly being exposed to programming in other instructional contexts. For example, one study found that there are as many jobs requiring advanced technology skills in industries outside of STEM as there are in STEM industries (*NAEP – 2014 Technology and Engineering Literacy*,

n.d.). Given this changing landscape, it is important to study programming in novel instructional contexts with populations of learners who not only lack prior programming experience but may also have never expressed an interest in learning programming to begin with.

Another limitation is the types of tasks used to study students' experiences learning programming. When researchers study students' cognitive and affective outcomes in introductory programming courses, they tend to use traditional programming tasks that involve writing and submitting code and define learning as students' ability to write code that runs correctly. However, writing and submitting code is only one instructional strategy that can be used to teach computer programming. Because writing and submitting code inherently focuses on programming outcomes and includes added element interactivity, it is unclear whether the cognitive and affective difficulties novice learners experience is due to the structure of the programming task, to the nature of learning programming itself, or to individual differences that students bring with them to the task of learning programming.

Further, since different programming task structures have different cognitive and affective affordances, they may elicit different patterns of emotions, motivations and beliefs. Traditional programming tasks which focus on producing correct code send implicit messages about the task goal and what counts as success when learning programming. Because students are rewarded for writing code that runs correctly, they may come to believe that writing correct code is what is important. Holding this belief may impact how students engage with programming tasks and how they interpret success and failure. For example, students may avoid trying new strategies for fear of failure. Holding the belief that learning programming is about writing correct code may also affect students' emotional responses to mistakes and challenges during learning. For example, if making a mistake prevents students from achieving the goal of submitting correct code, they may feel frustrated rather than curious about why the code didn't

work. By contrast, programming task structures that reward students for investigating, interpreting, and understanding why code works may elicit emotions such as curiosity and surprise, which activate deeper cognitive processing.

Finally, those studies that do investigate the effect of programming tasks for teaching computer programming tend to focus on cognitive, emotional, or motivational outcomes in isolation. For example, some studies have investigated the effect of worked examples on cognitive load when learning programming; others have investigated the effect of error messages on students' emotions. Few studies have investigated the effect of different instructional techniques on student cognitive, emotional, and motivational outcomes at once in one study.

In this dissertation, I argue that programming instruction could benefit from carefully designed tasks that consider the cognitive, emotional, and motivational challenges students face during a first encounter with computer programming. In the section that follows, I describe how predicting may be one such strategy that works through cognitive and affective processes to support positive learning outcomes for students first embarking on their computer programming journey.

Predicting as an Alternative Instructional Strategy

Asking students to predict what will happen prior to instruction is a popular teaching strategy that has been shown to support learning in a variety of domains (Brod, 2021). Reading teachers often ask students to predict what will happen next in a story or passage. Science instructors ask students to predict the outcomes of experiments. Yet the benefits of predicting have yet to be explored in the context of computer programming instruction. Below I discuss the research on predicting and why it might be particularly beneficial to students in the early stages of learning programming.

Predicting involves making a hypothesis about the outcome of a process or procedure. Unlike guessing, which is not based on prior knowledge, predicting involves drawing on some pre-existing knowledge followed by immediate feedback which can be used to compare the prediction to the actual outcome (Brod et al., 2018; Miller et al., 2013). In this way, predicting requires students to retrieve information and go beyond what is provided explicitly in instruction, placing it in a broader category of generative learning activities (Brod, 2021). However, predicting also has characteristics that set it apart from other generative learning strategies such as retrieval or elaboration. For example, predicting has been shown to activate emotions such as curiosity and surprise (when the prediction is not correct) which can direct attention during learning.

Support for benefits of predicting has been found across a number of academic contexts. For example, White and Gunstone (1992) discuss using a predict-observe-explain strategy to promote conceptual understanding in chemistry. The 'predict-observe-explain' strategy involves a three-part process in which students first predict the outcome of a physical experiment, then conduct the experiment and describe what they see, and finally, they reconcile any differences between the prediction and their observation. Predicting has also been shown to activate cognitive blueprints or plans that guide students' cognitive processing during reading (Blanton et al., 1990). For example, Nolan (1991) found that a combined strategy of self-questioning and prediction benefits reading comprehension among poor readers. They posit that one explanation for these outcomes is that predicting forces students to self-monitor their progress and thus increases their active cognitive processing of the information (Nolan, 1991). Predicting is also thought to increase motivation by increasing personal investment or involvement in the outcome (Nolan, 1991).

Despite the documented benefits of predicting in text comprehension, mathematics, and other complex domains, predicting has not directly been examined as a strategy for teaching computer programming. However, there are a number of reasons why predicting might be especially beneficial in this context, especially compared to traditional production activities in which students modify or write their own code. Table 1 shows a comparison of predicting and traditional production (code writing activities) and the mechanisms through which they might influence students' emotions, motivation, and learning outcomes. I discuss the differences between the two strategies in the section that follows.

Table 1. Predicting versus production activities

	Incentive structure	Motivation	Emotions	Cognitive processing
Modifying or writing code	Incentivizes performance	Belief that programming is about following steps, writing code to produce correct output; more fixed mindset towards programming; lower self-efficacy	Emotions tied to performance, frustration when code doesn't produce desired output; negative emotions in response to errors	Surface-level processing (following steps to produce correct output); increased cognitive load
Generating predictions	Incentivizes understanding	Belief that programming is	Emotions tied to understanding;	Deeper cognitive

about making mistakes and debugging; more growth mindset towards programming; greater self-efficacy	surprise in response to making an incorrect prediction; more positive emotions in response to error	processing of feedback and instruction

One way that that predicting may increase learning is by changing the incentive structure of the programming task. Whereas traditional programming activities focus on outcomes, predicting would make it impossible for students to define success as getting the correct answer. Instead, the goal of predicting would focus on understanding what the code does and why it works. Because predicting incentivizes understanding rather than getting the right answer, predicting tasks may activate different belief structures, goals, and approaches to learning. For example, students may adopt more of a mastery approach towards learning. As a result, students may engage in deeper approaches to learning, be less likely to attribute incorrect predictions to internal or stable traits, like intelligence, and experience fewer negative emotions during learning. Through these experiences, students also get in the habit of experiencing programming as a process of making and learning from predictions – which is a more accurate representation of what programmers actually do.

Another way that predicting might facilitate knowledge acquisition and more adaptive approaches to learning programming is by stimulating curiosity by creating a gap between what students know and do not know, which facilitates deeper cognitive processing of the material.

For example, predicting may surprise in response to incorrect responses, and thus increase students' attention to the code output, which in turn benefits learning (Brod et al., 2018).

A third way that predicting might benefit students' learning to program is by reducing the extraneous cognitive load that may be introduced by code writing activities. Previous research from the worked example literature suggests that providing students with opportunities to read worked out solutions of others can benefit learning in the early stages because it reduces the cognitive demands of learning and allows students to focus their attention on key elements of the problem (e.g., Sweller & Cooper, 1985). Making predictions also eliminates the added interactivity of writing code and reduces what students need to remember. For example, students only have to generate ideas about what the outcome will be; they do not have to remember correct programming syntax, or the steps required to produce the correct answer.

In sum, predicting activities have different cognitive, emotional, and motivational affordances than traditional code writing activities that may be more suited for the needs of beginning students. They 1) incentivize understanding over performance, which may facilitate more adaptive beliefs, goals, and approaches to learning as well as more positive emotions in response to setbacks and errors 2) facilitate deeper cognitive processing by directing students' attention to code comprehension and reducing the cognitive demands of the task, and 3) elicit positive, orienting emotions that facilitate comprehension, engagement, and promote more positive experiences and attitudes towards programming. But while there are anecdotal reports of teachers using predicting to teach programming, no studies have systematically investigated the effects of predicting as an alternative instructional strategy. This dissertation aims to bridge this gap.

Overview of the Present Research

Research Questions and General Design

Can changing the programming tasks in which students are first introduced to programming lead to more positive emotional trajectories, increased motivation, and increased learning? How do early experiences with programming relate to student outcomes? And finally, do programming tasks influence the effect of individual differences on emotion, motivation, and learning outcomes?

To answer these questions, undergraduate students were randomly assigned to one of two conditions – 1) a predict (Predict) condition in which participants were given instruction then asked to make and test predictions about pre-populated code, or 2) a traditional (Traditional) instruction condition in which participants were given instruction and then asked to write or manipulate code on their own.

Both conditions received the same explanatory text and instructional content. The only difference between the conditions was the task they were asked to perform in one condition students were asked to make predictions and run pre-provided code (Predict condition); in the other condition to write or manipulate code on their own (Traditional condition).

Hypotheses

Programming Task (Prediction vs. Production Condition)

I hypothesized that students assigned to the Predict condition would show more positive emotional trajectories, increased motivation, and increased learning than students assigned to the Traditional production-focused condition. This hypothesis was informed by prior research in which generating predictions led to increased curiosity and surprise – emotions that promote exploration and sensemaking. This hypothesis was further informed by research on computer programming which has found that students feel negative emotions in response to errors. Because predicting facilitates positive emotions such as curiosity and surprise while also shifting

focus away from performance, I expected students in the Predict condition to show more positive emotional trajectories, especially in response to challenges and setbacks.

In terms of motivation, I predicted that students assigned to the Predict condition would perceive the cost of learning programming to be lower and the value of programming to be higher, show increased self-efficacy towards programming, and be more likely to agree that they will take a programming course in the future. This prediction is based on social-cognitive theory and research, which suggests that learning environments can activate goals and belief structures that influence the goals students set and the way students respond to setbacks and errors during learning (Bandura, 1977 , 1986; Bandura et al., 2001; Schunk, 2012). Predicting tasks which focus on understanding over getting the correct answer will encourage students to adopt more mastery-oriented beliefs and approaches to learning, which will influence the way they perceive errors during learning. Predicting tasks may also reduce the cognitive burden of learning programming, which will lower students' perceptions of cost during learning – an important component of motivation.

With regard to learning, I predicted participants in the Predict condition would score higher on the learning assessment and generate more solutions when presented with a novel programming problem at the end of the experiment. This prediction is based on research that has highlighted the benefits of positive emotional experiences during learning as well as the research on students' cognitive processes when learning computer programming. Positive emotions can orient attention and facilitate exploration during learning. Predicting as an instructional strategy has also been shown to have cognitive benefits such as orienting students' attention to feedback, increasing engagement, and reducing extraneous cognitive load.

Learning Experiences and Interactions

The second research question asks whether the learning experiences and interactions in the Predict condition differ from those in the condition, and how those experiences and interactions relate to student outcomes. One difference may be that predicting reduces the cognitive demand and perceived cost of learning programming. Another difference may be that predicting directs student attention and increases engagement with learning materials. If that is true, then I would expect students in the Predict condition to rate the perceived extraneous cognitive load to be lower and the perceived germane cognitive load to be higher than students in the Traditional condition. I would also expect students in the Predict condition to show higher engagement during learning as evidenced by increased time spent and more words written on the end of chapter summaries.

The Role of Individual Differences

The third research question is exploratory in nature and asks how individual student characteristics relate to measures of emotion, motivation, and how these relationships might vary based on the type of programming task. I hypothesized that predicting would benefit all students and that these benefits would be enhanced for those students with poorer academic performance, those belonging to groups underrepresented in STEM, and those with lower initial motivation to learn programming.

Method

Participants

Participants were recruited from the Psychology subject pool and were awarded course credit for their participation. To avoid biasing participants' responses and to ensure recruitment of participants who may not be interested in computer programming, participants were informed that the goal of the research was to test a new online learning module, but not explicitly told that the task would involve programming. My goal was to have at least 60 participants in each

condition for a total of 120 students. However, I ran as many participants past 120 as was logistically possible to counteract the possible effects of attrition and potential technical difficulties.

In total, 166 participants signed up for the study. Participants who did not complete the experiment (N = 21), who completed the experiment in more than one sitting (N = 17), and who took less than 10 minutes to complete the experiment (N = 7) were excluded. The resulting analytic sample comprised 121 students, with 60 in the Predict condition and 61 in the Traditional condition. Of the 121 students, 67% were female (39 males, 1 prefer to self-describe), with an average age of 20.35 years old, who identified themselves as Asian/Asian American (36%), Black/African American (4%), Latino (17%), Middle Eastern/ North African (7%), More than one race/ ethnicity (13%), and White (22%) with an additional student who chose to self-describe as "Uzbek". Table 2 and Table 3 show a breakdown of sociodemographic characteristics across the two conditions. Chi-square analysis revealed no significant associations between condition and gender ($X^2(2) = 4.56$, $p = .1$), condition and race/ethnicity ($X^2(2) = 6.38$, $p = .382$, and condition and prior experience with R ($X^2(2) = 0.33$, $p = .6$).

Table 2. Sociodemographic characteristics of the sample.

	Condition	
	Predict N (%)	Traditional N (%)
Total	60 (100%)	61 (100%)
Gender		
Female	35 (58%)	46 (75%)
Male	24 (40%)	15 (25%)

Non-binary / prefer to self-describe	1 (2%)	0 (0%)
Race/ethnicity		
Asian/Asian Am.	20 (33%)	23 (38%)
Black/African Am.	1 (2%)	4 (7%)
Latino	10 (17%)	11 (18%)
Middle Eastern/ North African	6 (10%)	2 (3%)
Multi-racial	10 (17%)	6 (10%)
Prefer to self-describe	1 (2%)	0 (0%)
White	12 (20%)	15 (25%)
Familiar with R		
Yes	14 (23%)	17 (28%)
No	46 (77%)	44 (72%)

Table 3. Descriptive statistics for Age and GPA .

	Condition		*F*	df1, df2	*p*
	Predict M (SD)	Traditional M (SD)			
Age	20.02 (1.82)	20.68 (3.11)	2.06	1,118	.2
GPA	3.15 (1.36)	3.44 (0.86)	1.91	1,118	.2

Materials

Development of the Learning Materials

The learning materials consisted of four modules and included a total of seventeen brief activities. There are a variety of topics that might be covered in an introductory course in computer programming and a variety of ways to introduce them. To guide the design of the modules, I used the following criteria: First, the modules should teach topics typical of a self-paced introductory programming course, such as basic syntax, assignment, data structures, and

operators. These are also topics with which students struggle. Second, the modules must include explanatory text interleaved with questions and interactive coding activities. These interactions are typical of what students would encounter in an online course. Third, each module must take between ten to fifteen minutes to complete. The purpose of limiting the length of the modules was to avoid overwhelming participants. This length is also typical of what students might experience in self-paced online courses. Finally, the modules must be challenging, but accessible for students with no programming experience and limited knowledge of mathematics. A description of the learning modules and level of difficulty is shown in Table 4.

Table 4. Content of the learning modules

Module name	Activities	Level of difficulty
1: The print() function	4	Very easy
2: Arithmetic operators	3	Easy
3: Objects	4	Medium
4: Vectors	6	Hard

Creating the Experiment in Qualtrics

The experiment was created and administered in the research software, Qualtrics. Qualtrics allows researchers to randomly assign participants to different conditions and embed custom content in surveys. Qualtrics also provides tools for managing participants and can be integrated into participant recruitment systems.

Adding the Interactive Programming Content

To facilitate realistic programming interactions, I used Datacamp Light (Datacamp Light v2.3.0, 2020), an interactive coding widget for R and python. Datacamp Light lets users develop their own coding exercises which can be embedded as html or markup in websites. To embed the

Datacamp Light exercises in Qualtrics, I created an html file for each exercise, then embedded the files as iframes within blank Qualtrics questions.

Since I used an iframe, I wasn't able to capture participants' interactions with the coding exercises. To gain some sense of what participants did when coding, I included a follow-up question after each coding window (e.g., "Which output did you see?") that prompted students to select the output they saw from five different options. The first four options were presented in a random order and the final option was always "I saw something else". If students selected "I saw something else," they were prompted to type the output they saw (see Figure 1 below).

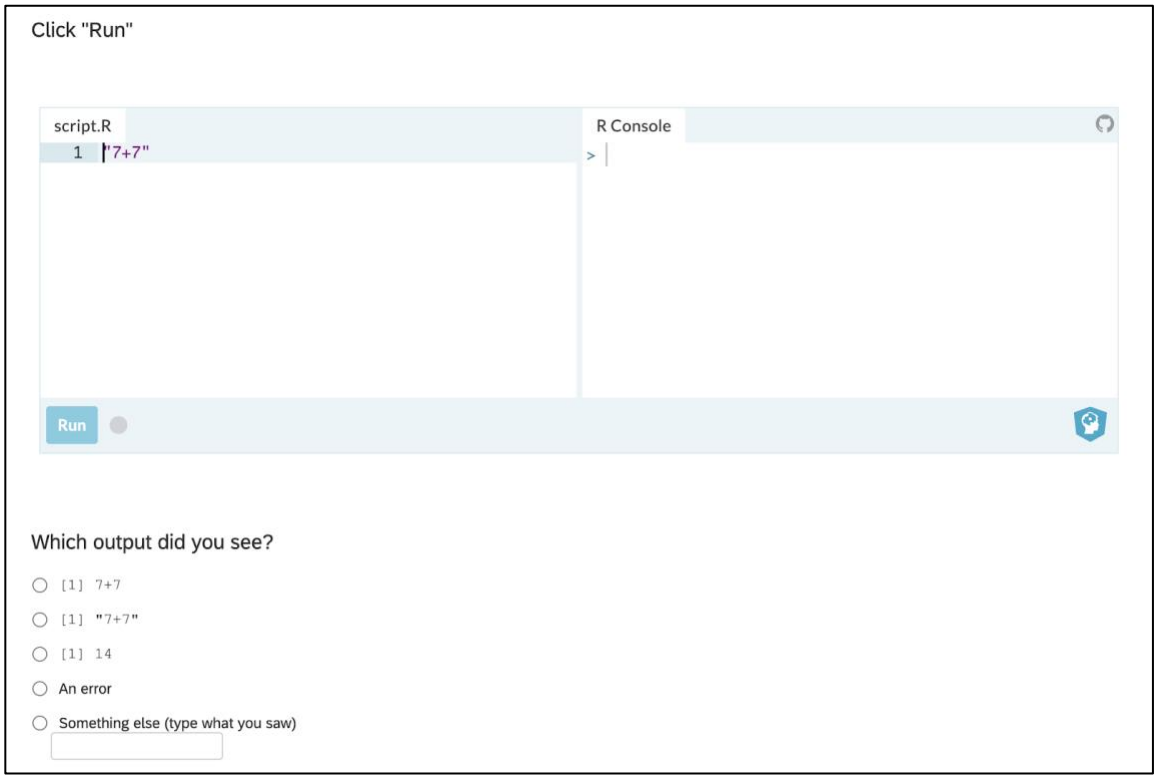

Figure 1. Example of how students recorded output after running code in the interactive coding window.

Though Datacamp Light offers the ability to test code for correctness and provide correct/incorrect feedback, I opted not to use this feature. Prior research has shown that feedback influences students' emotions when learning to program (Maricuţoiu, 2006, Traver, 2010) and even minor variations in how feedback is presented can influence students' perceptions. Thus, to ensure that any differences in students' experiences were due to the nature of the programming interactions rather than feedback on correctness, I opted to remove feedback messages on correctness from the programming environments in both conditions. Students, regardless of condition, were provided feedback in the form of the correct output on the page following each activity along with instructional text explaining the code and output (see Figure 2). The correct output and subsequent explanatory text were the same for students in the Predict and Traditional conditions.

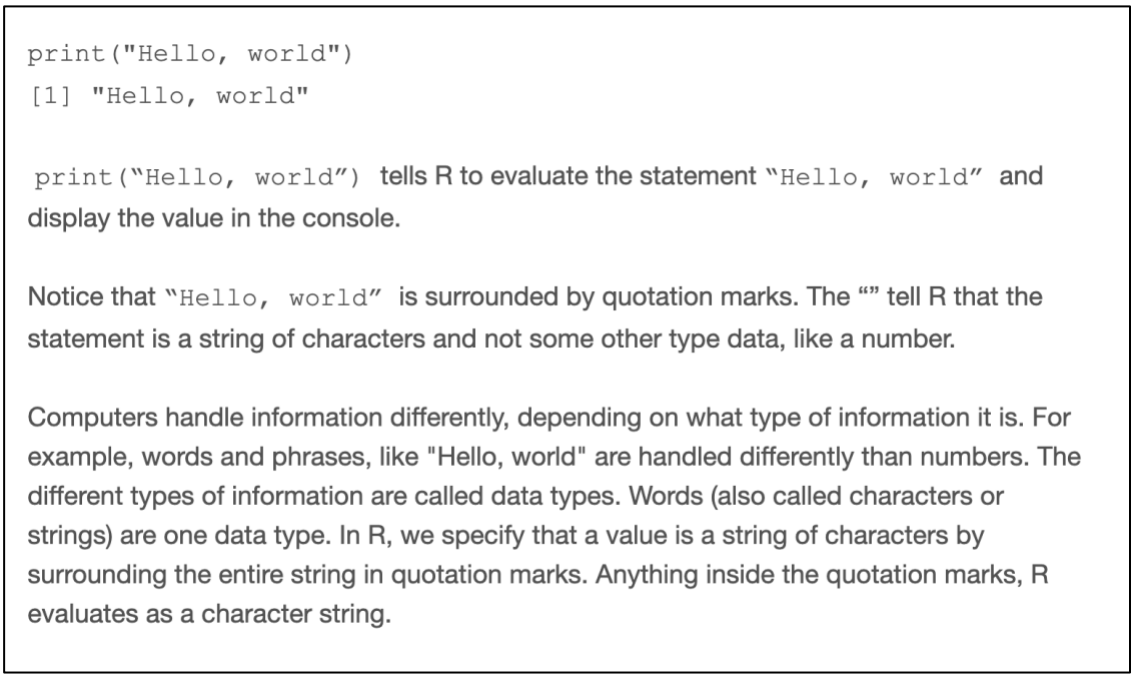

Figure 2. Feedback in the form of example code and output followed by explanatory text. Example code and explanatory text were the same across the Predict and Traditional conditions.

Procedure

Upon signing up for the study, participants received a link to access the experiment and were randomly assigned to either the Predict or Traditional instructional conditions. All participants began by reading an overview of the experiment, confirming their eligibility, and filling out a questionnaire with measures of motivation (perceived cost of the activity, goals, sentiment towards R, attitudes towards computers, need for cognition, and programming mindset).

Next, participants worked their way through a series of four instructional modules. Participants assigned to the Predict condition were first provided some introductory information, shown some R code, and asked to make a prediction by selecting from one of five multiple choice options (i.e., "When we click Run, which output do you think we will see?"). On a separate page, students were reminded of their prediction and prompted to run pre-provided code in an embedded programming window. Students ran the code and then reported their results by selecting from one of five multiple choice options. On the next page, students were provided explanatory text that explained the R code and why it worked.

Participants assigned to the Traditional condition were provided introductory information and example code. They were asked to modify the code that was provided, click "Run", and then report the output they saw (i.e., "Which output did you see?"). The multiple-choice options were the same as the options in the Predict condition. On the next page, students were provided an explanation of the correct R code and why it worked. Examples of how an activity appeared in the Predict and Traditional conditions are provided in Figure 3.

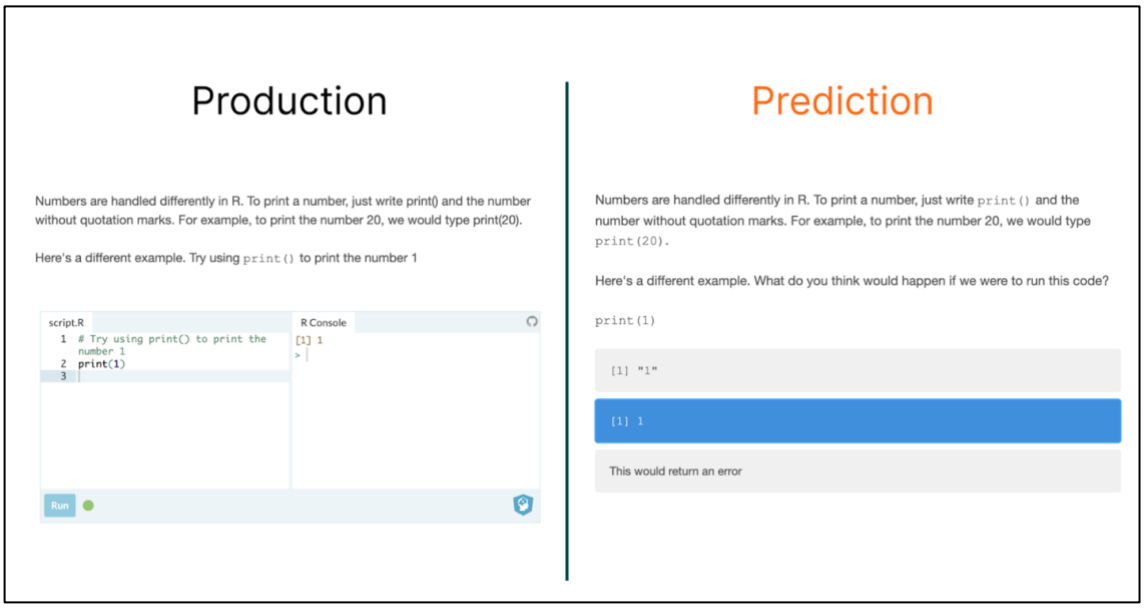

Figure 3. Comparison of instruction in the Traditional (production) condition and the Prediction condition.

At the end of each module, participants in both conditions rated their emotions and wrote a brief summary of what they had learned. After completing the instructional part of the experiment, participants were asked to complete a summative assessment of their learning. Finally, at the end of the session participants completed a post survey with measures of motivation, sentiment towards programming, programming mindset, math background, and demographic information. See Figure 4 for an illustration of the study procedure as well as a timeline of when various measures were administered.

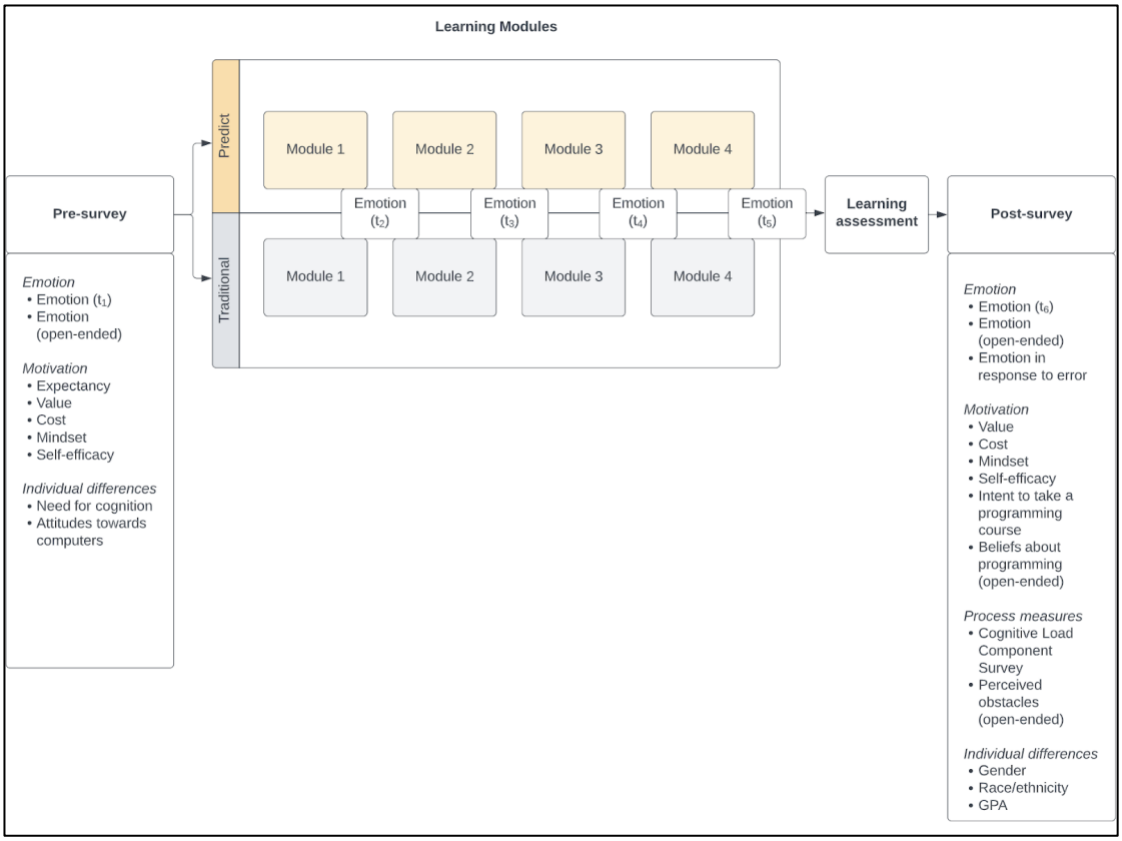

Figure 3. Experimental procedure. Sentiment towards R was measured at six timepoints ($t_1 - t_6$).

Measures

An overview of the measures in this study is shown in Table 5.

Table 5. Overview of the study measures.

Outcome measures
Emotion 1. Sentiment towards R(t_1-t_6) 2. Emotion in response to error Motivation 1. Expectancy 2. Value 3. Cost 4. Self-efficacy

5. Mindset
6. Intent to take programming course
7. Beliefs about computer programming (open-ended)

Learning
1. Score on post-lesson assessment
2. Number of correct solutions on novel programming problem

Process measures

Behavior
1. Word count
2. Duration (s)
3. Performance on lesson activities

Cognitive Load Component Survey
1. Intrinsic load
2. Extraneous load
3. Germane load

Perceptions
1. Perceived obstacles to success (open-ended)

Individual difference measures

Cognitive dispositions
1. Need for cognition
2. Attitudes towards computers

Sociodemographic characteristics
1. Gender
2. Race/ethnicity
3. GPA

Outcome Measures

Outcome measures included measures of emotion, motivation (including beliefs), and learning.

Emotion. Two measures of emotion were included in this study.

Sentiment Expressed Towards R during Learning. Sentiment is defined as a thought about an object that is prompted by emotion (Munezero et al., 2014). Participants used a slider to rate their sentiment expressed towards R at six time points throughout the study on a continuous

scale from -100 (Extremely negative) to +100 (Extremely positive). The time points were: (t_1) prior to starting the activity on the pre-survey, (t_2 - t_5) immediately, after each of the 4 learning modules, and (t_6) on the post-survey after the transfer task. Participants were prompted to explain the rationale for their ratings at timepoints1(presurvey) and time point 6 (post-survey). The purpose of these qualitative measures was to add context to students' sentiment ratings and facilitate deeper insights into their experiences.

***Emotion in Response to Error*.** To test whether participants who generate predictions experience reduced negative emotion in response to failure than students who produce code themselves, I measured participants' anticipated response to a hypothetical error scenario. At the end of the experiment, on the post-survey, participants were shown a screenshot of incorrect R code and the associated error message. They were prompted to imagine they had written the R code and received the error message shown. They were then asked to rate how they felt using a slider on a scale from -100 (Extremely negative) to +100 (Extremely positive).

Motivation. I measured several factors known to influence motivation: expectancy, value, and cost appraisals, motivational beliefs (programming mindsets and self-efficacy), intent to take a programming course in the future, and general beliefs about programming.

Expectancy, Value, and Cost Appraisals. Expectancy, value, and cost appraisals were measured using items adapted from Kosovich et al. (2015) and Gaspard et al. (2017). Value and cost are useful predictors of students' academic motivation and achievement in STEM domains (Jiang et al., 2018) and can be influenced by educational materials (Barron & Hulleman, 2015). *Value* items addressed intrinsic task value, utility value, and intrinsic motivation (examples include "I think this activity will be/was interesting", "What I am going to learn/learned in this activity is useful", and "What I'm going to learn/ what I learned in this activity will be relevant to my everyday life."). *Cost* was measured on the pre-survey and on the

post-survey using the following four items: “For some reason, computer programming seems like it will be particularly hard for me.”, “I think I would have to give up too much to learn programming”, “I think learning computer programming would be too stressful for me”, and “I think learning computer programming would take up too much of my time.” Students responded to all items using a six-point scale from “Strongly disagree” to “Strongly agree.”

Programming Mindset. Programming mindset refers to the belief that programming ability is either malleable – something that improves with practice – or an innate ability that can’t be changed. Programming mindset was measured at two time points: on the pre-survey (t_1) and on the post-survey (t_6) using items adapted from the three-item Growth Mindset Scale (Dweck, 1999) (examples include “anyone has the ability to learn programming and be good at it” and “you can learn new things, but you can’t really change your programming ability”). Participants rated their agreement with each statement using a six-point scale from “Strongly disagree” to “Strongly agree”. Higher scores indicated more of a growth mindset toward computer programming. This measure was used to test whether the instructional condition (Predict vs. Traditional) influenced students’ beliefs about computer programming. It was also used to account for variations in participants’ beliefs at the start of the activity.

Self-efficacy. Self-efficacy was measured at two time points: on the presurvey (t_1) and on the post-survey (t_6). Participants rated their agreement with two statements (e.g., “I think that I could learn computer programming”, “I think I could handle the more difficult programming problems.”) using a six-point scale from “Strongly disagree” to “Strongly agree”.

Intent to Take a Programming Course in the Future. Intent to take a programming course was measured using a single item on the post-survey: “How likely are you to take a programming course?” Students rated their agreement with the statement on a ten-point scale from “Not at all likely” to “Extremely likely”. I also included an additional question in which

students were asked to indicate (yes or no) whether or not they would be willing to help the researchers test other programming activities in the future.

Beliefs About Computer Programming. On the post-survey at t_6, students were asked to "write down all the words that come to mind when you think about computer programming." This measure was added to capture differences in participants' beliefs about what programming is and what learning to program requires.

Learning. Learning was measured in two ways: score on a post-lesson assessment of learning and number of correct solutions produced on a novel programming task.

Score on the Learning Assessment. After the four learning modules and before the post-survey, participants completed a sixteen-item assessment that covered concepts related to each of the four learning modules. Participants were awarded one point for each correct item, yielding a total correct score between 0 and 16, with higher scores indicating greater learning.

Number of Correct Solutions on a Novel Programming Task. After completing the learning assessment but before the post-survey, participants were presented with a novel R coding task and asked to generate as many solutions as possible. I counted the number of correct solutions generated by each participant and used the total score as an additional measure of learning. This measure is similar to measures of divergent thinking, an element of creative problem-solving (Guilford, 1967; Kwon et al., 2006).

Process Measures

Process measures were included to understand how participants interacted with and perceived the learning activities.

Duration (in seconds). Duration – calculated as the total amount of time participants spent from first clicking the link to open the survey to clicking "submit" at the end of the survey –was captured automatically by the survey software. I included this measure to account for

differences in the amount of time spent on the activity and confirm that the total time required did not differ between the Predict and Traditional conditions.

Word Count. As a proxy for engagement, I calculated the total number of words written on the four summary questions at the end of each module. I included word count to account for differences in the number of words students produced during learning, as generative activities, such as writing, in themselves may facilitate learning.

Performance. Feedback on performance during learning can influence motivation, emotions, and learning outcomes. To account for this potential confound, I calculated performance as the ratio of the number of correct responses on the activities in the learning modules. For students in the Predict condition, responses were marked correct if the prediction matched the pre-provided code's output. For students in the Traditional condition, responses were marked correct if the output they produced matched the expected output for the activity. I included a measure of performance to account for variations in feedback (either positive or negative) that may have resulted from students' interactions and to ensure that one condition did not elicit more "correct" responses than the other.

Cognitive Load Component Survey. Nine items adapted from the Cognitive Load Component Survey (Morrison, Dorn, & Guzdial, 2014) were included in the post-survey to measure participants' perceptions of the activity they just completed. Participants used a scale of 0 to 10 to estimate the mental resources used:

- to understand the content – intrinsic cognitive load (i.e., "The topics covered in the activity were very complex."),
- to make sense of the instructional materials – extraneous cognitive load (i.e., "The instructions and/or explanations during the activity were very unclear."),

- for comprehending and remembering the material – germane cognitive load (i.e., "The activity really enhanced my knowledge and understanding of computing/programming.").

I included these component measures to compare students' perceptions of the materials and activities across the two conditions. Specifically, I wanted to ensure that the intrinsic cognitive load did not differ across the two conditions (since the content was the same).

Obstacles (open-ended). To gain a deeper understanding of students' experiences with and perceptions of the learning activity, I included an open-ended question, "What obstacles, if any, did you encounter when completing this activity?" This item helped to detect issues with the experimental materials and any differences in students' experiences not captured by rating scales.

Individual Difference Measures

Prior research has identified individual characteristics that may influence students' experiences and outcomes when learning computer programming. I included the following individual difference measures:

Attitudes Towards Computers. People hold different attitudes towards computers which may influence their experiences and outcomes when learning computer programming. To account for these differences, I included seven items to assess students' attitudes towards computers. Six items addressed how much students liked computers (e.g., "I enjoy working with and learning about computers.", "I would like to learn more about computers.", "I like the idea of taking computer courses", "I find working with computers enjoyable.", "Computers just don't appeal to me" (reverse-coded), "I like learning about computers") and a seventh item addressed students' computer-related anxiety ("In general, I feel anxious around computers"). After reverse coding negatively phrased items, I created a composite variable by averaging the seven items.

Need for Cognition. Need for Cognition is considered a personality trait that describes an individuals' tendency to seek out and enjoy solving complex problems (Cacioppo et al., 1984). Need for cognition has been linked to openness to experience, curiosity, and creative thinking, attributes which may influence students' learning and engagement in complex and unfamiliar domains like computer programming. People with a high need for cognition tend to engage in thinking about complex topics, enjoy thinking, and are motivated to apply thinking skills with little prompting. To capture individual differences in Need for Cognition, I adapted six items from the revised Need for Cognition Scale (Cacioppo et al., 1984) averaged to create a composite need for cognition variable, with higher scores indicating higher need for cognition.

Sociodemographic Variables. Participants reported demographic information such as age, gender, and race/ethnicity at the end of the activity on the post-survey. I included these measures to characterize the sample and to test whether the effect of condition (Predict vs. Traditional) varied for students of different backgrounds. As a measure of academic performance, participants self-reported their GPA to the nearest tenth of a point using a sliding scale from 0.00 to 4.00. The purpose of this measure was to capture differences in participants' general academic performance, as participants who perform better academically may be more likely to perform better on the activities in this study.

Data Analysis Plan

Data were analyzed using quantitative approaches with qualitative insights included to extend quantitative findings. To characterize the sample, I began by exploring distributions of the pre-survey measures and their correlations. I then used *t*-tests to confirm random assignment was successful, evidenced by no detectable differences in the pre-survey measures by condition. For all analyses, I used R version 3.6.2 (R Core Team, 2019).

Group Differences in Learning, Motivation, and Emotion

The primary research question was whether students assigned to the Predict condition differed from students assigned to the Traditional condition in terms of their emotions, beliefs, motivation, and learning outcomes. To test whether students in the Predict condition demonstrated increased motivation, interest, and learning compared to students in the Traditional condition, I used multiple regression. Separate analyses were conducted for each outcome variable. The independent variable was whether the participant was assigned to the Predict or Traditional condition. For each of these analyses, I fit two models: a simple model with condition as the sole independent variable and a complex model with gender, GPA, and need for cognition as covariates. Gender, GPA, and need for cognition were included because they have each been shown to individually affect the outcome variables in this study.

Process Measures and their Relation to Learning, Motivation, and Emotion

The second research question was whether students in the Predict and Traditional conditions differed in terms of their interactions with and perceptions of the learning materials and how those process measures, in turn, related to learning, motivation, and emotion. First, I analyzed the distribution of process measures and used *t*-tests to test for group differences. I then used multiple regression analysis to test the effect of process measures on emotions, motivation, and learning at the end of the experiment.

The role of Individual Differences

Given the relatively small sample size, this study was not sufficiently powered to statistically test interactions between condition and individual difference variables on student outcomes. Thus, to investigate the third research question – whether the effect of individual differences in students' backgrounds, beliefs, and initial motivation on emotions, motivation, and learning varies based on the instructional condition (Predict versus Traditional), I conducted exploratory analyses to visually compare how the relationship between individual difference

variables and student outcomes might vary as a function of the programming task (Predict versus. Traditional).

Results

Characteristics of the Sample

Descriptive statistics for all pre-survey measures, broken down by condition, and results from the t-tests are displayed in Table 6. Across the two conditions, students did not differ in their initial emotion, programming mindset, perceived cost of learning programming, perceived value of programming, initial expectations for success, initial self-efficacy for learning computer programming, attitudes towards computers, or need for cognition. These findings suggest that the randomization worked as intended.

Table 6. Descriptive statistics and t-tests comparing pre-survey measures between the Predict and Traditional instructional conditions.

	Predict *M*(SD)	Traditional *M*(SD)	*b*1 [95% CI]	*t*(df)	*p*
Emotion					
Sentiment (t1)	7.65(49.77)	-0.26(50.97)	7.91 [-10.22, 26.05]	-0.86(118.99)	.4
Motivation					
Mindset (t1)	3.84(0.86)	3.8(0.89)	0.04 [-0.28, 0.35]	-0.24(118.94)	.8
Cost (t1)	3.54(0.99)	3.75(1.1)	-0.21 [-0.59, 0.16]	1.13(118.2)	.3
Expectancy	3.86(0.93)	3.74(1.15)	0.13 [-0.25, 0.51]	-0.68(114.58)	.5
Value	4.13(1.03)	4.06(1.12)	0.06 [-0.32, 0.45]	-0.35(118.39)	.7
Self-efficacy (t1)	3.86(1.11)	3.70(1.141)	0.15	0.75(118.98)	.5

			[-0.25,0.56]		
Individual difference variables					
Attitudes towards computers	4.12(0.98)	3.88(1.04)	0.24 [-0.12, 0.6]	-1.32(118.82)	.2
Need for cognition	3.44(0.74)	3.41(0.73)	0.03 [-0.23,0.30]	0.24(118.9)	.8
GPA					

To understand the relationship between measures of participant characteristics at the start of the study, I ran pairwise correlations between pre-survey measures of emotion, motivation (expectancy, value, cost, programming mindset, self-efficacy), and individual difference variables (attitudes towards computers, Need for Cognition, GPA).

Figure 4 shows a correlation plot with correlation coefficients for each pairwise comparison. Students who held more positive attitudes towards computers perceived the cost of learning programming to be lower, held higher expectations for success, valued programming more, showed more growth-oriented mindset towards programming, and held higher self-efficacy for their ability to learn computer programming. Need for cognition was also positively correlated with measures of emotion and motivation. Students who scored higher on need for cognition were more likely to show positive attitudes towards computers, rate their self-efficacy for learning programming higher, demonstrate more positive emotions, hold higher expectations for success, value learning programming more, and show more growth mindsets towards learning programming. Measures of motivation were related to one another as would be expected based on prior literature. GPA was positively correlated with cost of learning computer programming but not with any of the other pre-survey variables.

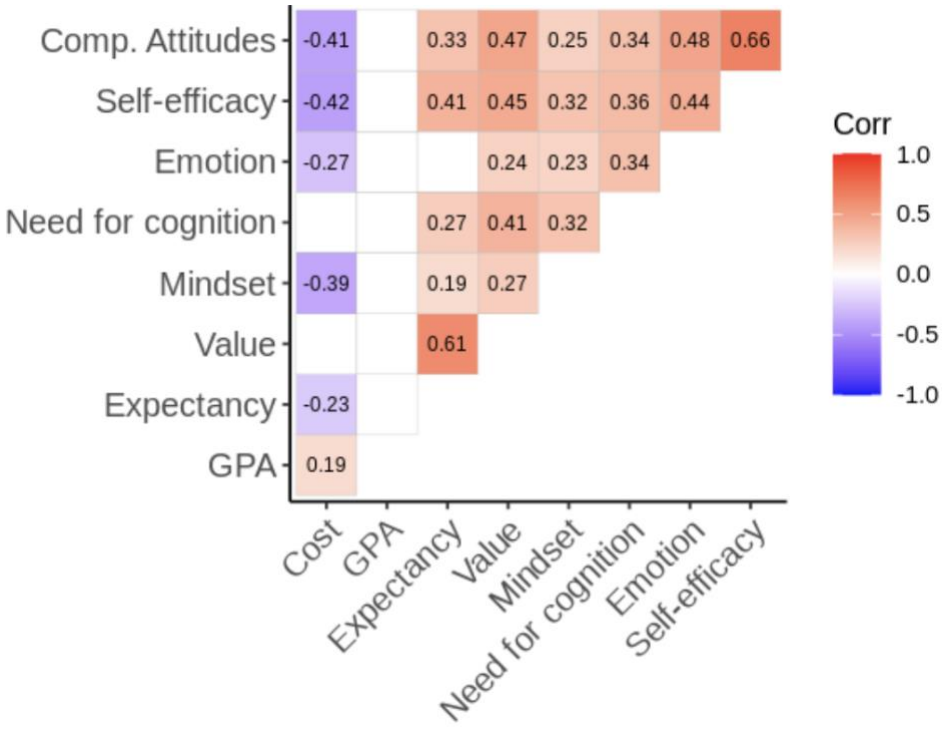

Figure 4. The pairwise correlation plot for pre-survey variables. The numbers show the correlation coefficients. Warm colors indicate positive correlations. Cool colors indicate negative correlations. Non-significant correlations are blanked out.

Effect of Condition on Student Outcomes

The Effect of Condition on Emotion

As a reminder, students expressed their sentiment towards R by moving a slider on a continuous scale from -100 (extremely negative) to +100 (extremely positive) at six timepoints. Participants used the same slider and scale to rate how positive or negative they felt in response to a hypothetical programming error on the post-survey.

Table 7 shows the distribution of sentiment ratings at each time point for students in the Predict and Traditional conditions. Though students in the two conditions did not differ

significantly prior to the start of the activity (t_1) or after the first two modules (t_2-t_3), students in the Predict condition demonstrated more positive sentiment after modules 3, 4, and 5 (t_4-t_5) and on the post-activity survey (t_6).

Table 7. Descriptive statistics and t-tests comparing sentiment ratings between the Predict and Traditional instructional conditions at each of the six time points.

Time	Predict *M*(SD)	Traditional *M*(SD)	b_1 [95% CI]	*t*(df)	*p*
1	7.65(49.78)	-0.26(50.97)	7.91[-10.22, 26.05]	-0.86(118.99)	.4
2	46.75(43.60)	30.93(51.53)	15.82[-1.38, 33.01]	-1.82(116.41)	.07
3	48.88(40.70)	31.66(50.25)	17.23[0.75, 33.70]	-2.07(114.79)	.04
4	53.96(43.55)	15.13(56.55)	38.84 [20.64, 57.02]	-4.24(112.56)	<.0001
5	46.91(46.57)	6.54(60.81)	40.38[20.86, 59.90]	-4.10(112.29)	<.0001
6	53.60(46.11)	25.75(59.50)	27.85[8.66, 47.03]	-2.88(112.86)	.005

On the post-survey, participants were shown a screenshot of negative feedback on a programming task similar to the ones they completed and rated how the feedback made them feel on a scale of -100 (negative) to +100 (positive). The distribution of emotion ratings for the overall sample is shown in Figure 5. Most students felt neutral or slightly negative (M = -4.24, SD = 50.79), but responses ranged from -100 to 100.

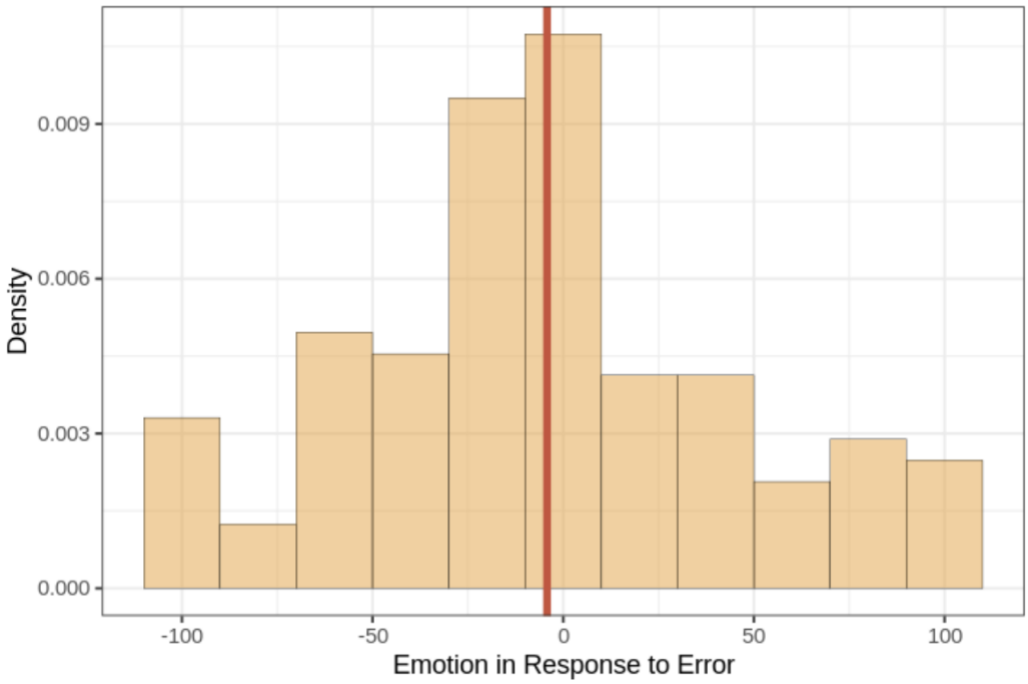

Figure 5. Histogram of participants' emotion ratings when shown a hypothetical programming error. Vertical red line represents the mean.

Figure 6 shows the distribution of emotion ratings to the hypothetical error scenario for students in the Predict and Traditional conditions. Participants in the Predict condition rated their emotion as neutral, whereas participants in the Traditional condition rated their emotion as negative (b_1= 28.77 , F(1, 119)=10.30, p =.002, *95% CI*[11.02, 46.52]). The effect of condition on emotion in response to error remained statistically meaningful when GPA, need for cognition,

and emotion rating on the pre-survey were included as covariates in the model (b_i= 27.24 , F(1, 114)=9.63, p =.003, *95% CI*[9.42, 45.06]).

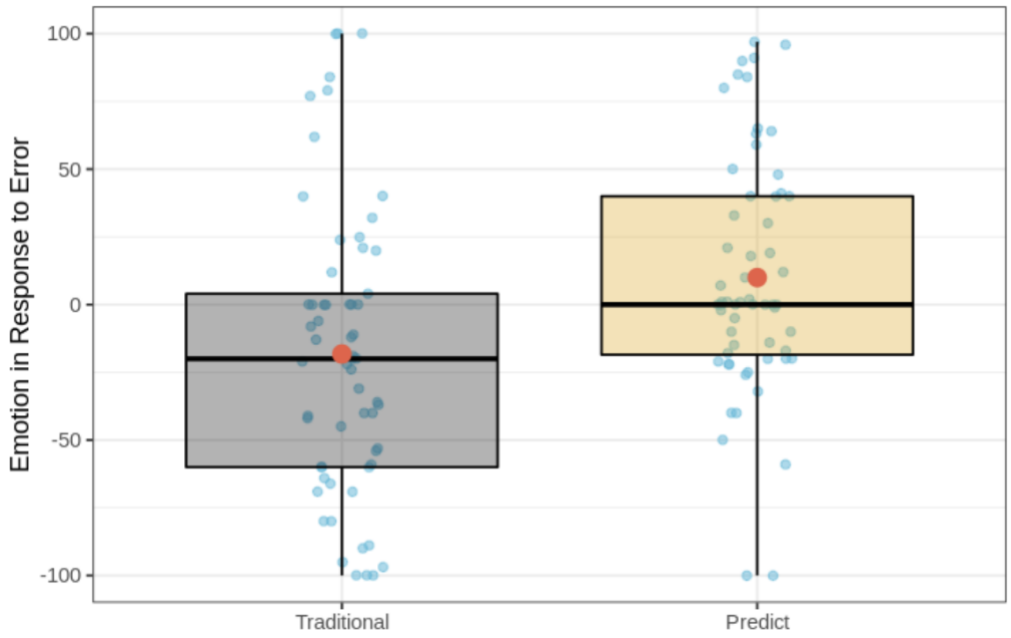

Figure 6. Boxplot of the effect of instructional condition on emotion rating in response to a hypothetical error. The points represent the individual participants' scores. The horizontal line represents the group medians. The red dots represent the means.

Students' open-ended explanations provide some insight into what features of instruction may have influenced their emotions during learning. For example, students in the Predict condition may have experienced positive emotions when their experience did not match negative perceptions. In the words of one student in the Predict condition, "I thought that coding with R was very fun to learn. I was very nervous to learn R at first, but it was not as difficult as I thought." In the words of another student assigned to the Predict condition, "It was really fun to learn R this way. I thought even the basics of coding would be too intimidating, but this was

quite cool, and I'd love to learn more about it. I feel even more positive than before about R.' Another student in the Predict condition said, "I think I would be able to handle R with some effort. It makes me feel more positive than facing something unknown."

For students in the Traditional condition, issues with remembering programming syntax may have contributed to their more negative emotional experiences. According to one student, "It was not as bad as I expected, but it does get a little confusing and stressful to remember all the commands. I don't really like how sensitive R is with commands. This definitely isn't for me." Others in the Traditional condition explicitly mentioned feeling discouraged when their code didn't work (i.e., "I feel like I could only follow along for a period of time, before I got confused. It was discouraging to me when I did not get a correct answer because I was focused and thorough.") while others felt positive when their code ran correctly (i.e., "I thought it was fun to type and have the code return the answers I was expecting, it would be helpful for more complex situations but I don't feel the code I have learned will be applicable."). Though only anecdotal examples, these responses suggest that, for some students, writing and submitting code may have contributed to their emotional experience during the activity.

The Effect of Condition on Motivation

Descriptive statistics for measure of motivation on the post-survey are shown in Table 8. Participants randomly assigned to the Predict condition did not differ from students randomly assigned to the Traditional condition in terms of their perceived value of computer programming, programming mindset, or self-efficacy beliefs, but they did differ from participants in the Traditional condition in terms of the perceived cost of learning programming and their intent to take a programming course in the future.

Table 8. Descriptive statistics and t-tests comparing post-survey measures of motivation between the Predict and Traditional instructional conditions.

Time	Predict *M*(SD)	Traditional *M*(SD)	b_1 [95% CI]	*t*(df)	*p*
Value	4.52(1.30)	4.30(1.34)	0.23 [-0.25, 0.70]	0.95(118.97)	.3
Cost	2.94(1.09)	3.51(1.34)	-0.58 [-1.02,-0.14]	2.61(115.15)	.01
Intent to take programming courses	6.30(3.09)	5.18(3.01)	1.12 [0.02,2.22]	2.02(118.78)	.05
Programming mindset	4.09(1.06)	3.84(0.95)	0.25 [-0.12,0.61]	1.34(117.33)	.2
Self-efficacy	4.20(1.40)	3.92(1.31)	0.28 [-0.21,0.77]	1.14(118.23)	.3

Figure 7 shows the distribution of cost ratings on the post-survey. On average, participants rated the cost of learning computer programming a 3.22 out of 6 with a standard deviation of 1.25.

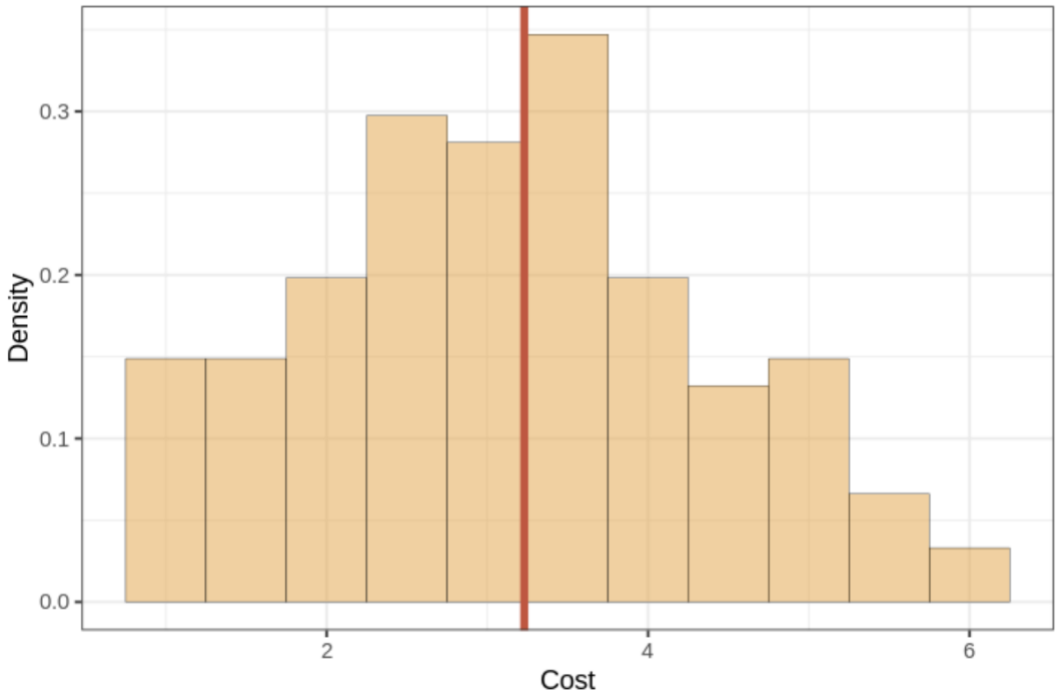

Figure 7. Histogram of participants' cost ratings on the post-survey. Vertical red line represents the mean.

Figure 8 compares post-survey cost ratings for participants in the two conditions. Participants in the Predict condition perceived learning programming to be less costly than participants in the Traditional condition (b_1= -0.57, $F(1, 119)=6.80$, p = .01, *95% CI*[-1.0, -0.13]). The effect of condition on post-survey cost ratings remained statistically meaningful when GPA, need for cognition, and pre-survey cost ratings were included in the model (b_1= -0.35, $F(1, 114)=5.39$, p = .02, *95% CI*[-0.66,-0.05].

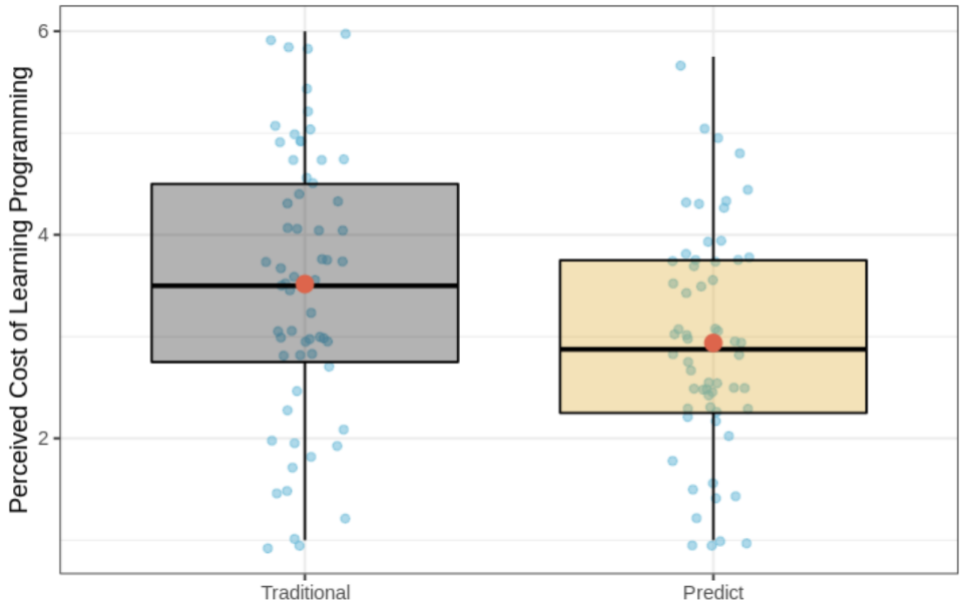

Figure 8. Boxplot of the effect of instructional condition on the average rating of cost on the post-survey . Points represent the individual participants' scores. The horizontal line represents the median. The red dot represents the mean.

Figure 9 shows the distribution of students' ratings of how likely they would be to take a programming course in the future. On average, students rated their likelihood of taking a programming course a 5.73 out of 10 with a standard deviation of 3.09, indicating they were only moderately likely to take a programming course in the future.

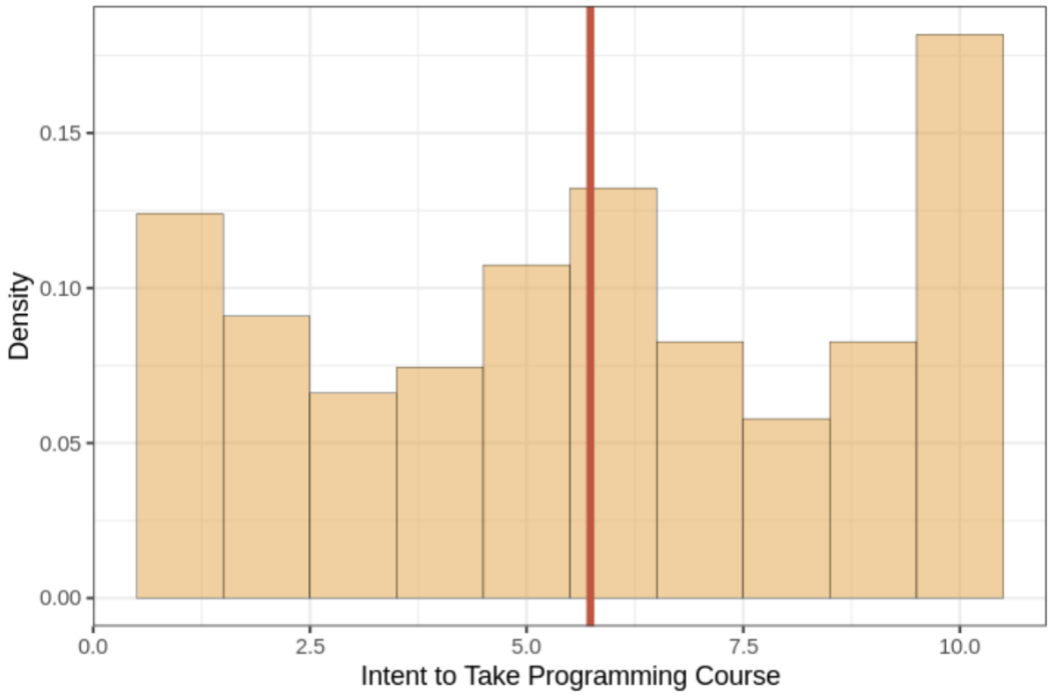

Figure 9. Histogram of intent to take a programming course. The vertical red line represents the mean.

Figure 10 compares how likely students would be to take a future programming course for students in the two conditions. On average, participants in the Predict condition showed stronger agreement with a statement about their intention to take a programming course in the future (b_i= 1.12 , F(1, 119)=4.08, p =.05, *95% CI*[0.02, 2.21]). The effect of condition on intent to take a programming course in the future remained statistically meaningful when GPA and Need for Cognition were included as covariates in the model (b_i= 1.16 , F(1, 116)=4.46, p =.04, *95% CI*[0.07, 2.24]).

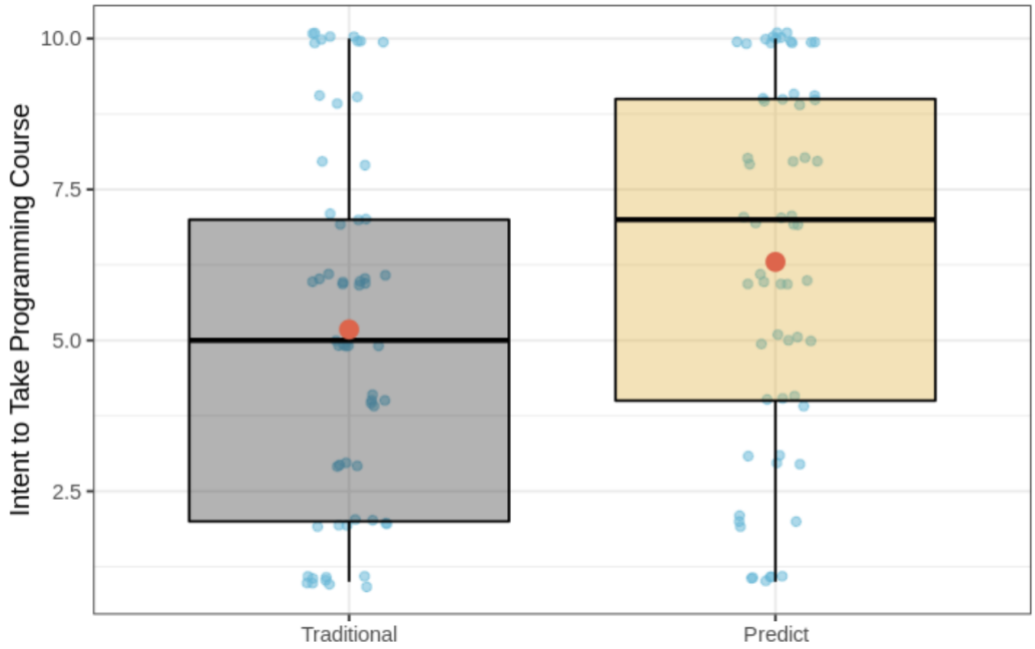

Figure 10. Boxplot of the effect of instructional condition on the average intent to take a programming course. Points represent the individual participants' scores. The horizontal line represents the median. Red dots represent the mean.

The Effect of Condition on Learning

The distribution of scores on the post-lesson learning assessment and the distribution of the number of correct solutions generated on a novel programming task are shown in Figures 11 and 12, respectively. On average, participants answered 9.21 out of 16 questions correctly with a standard deviation of 3.86.

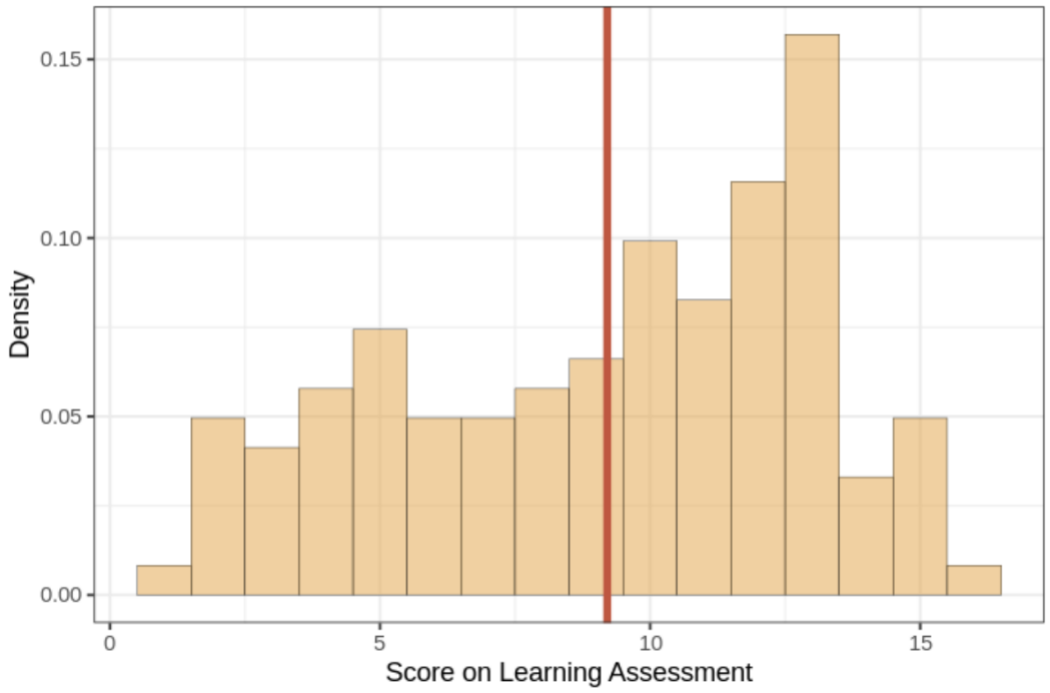

Figure 11. Histogram of scores on the post-lesson assessment. The vertical red line represents the mean.

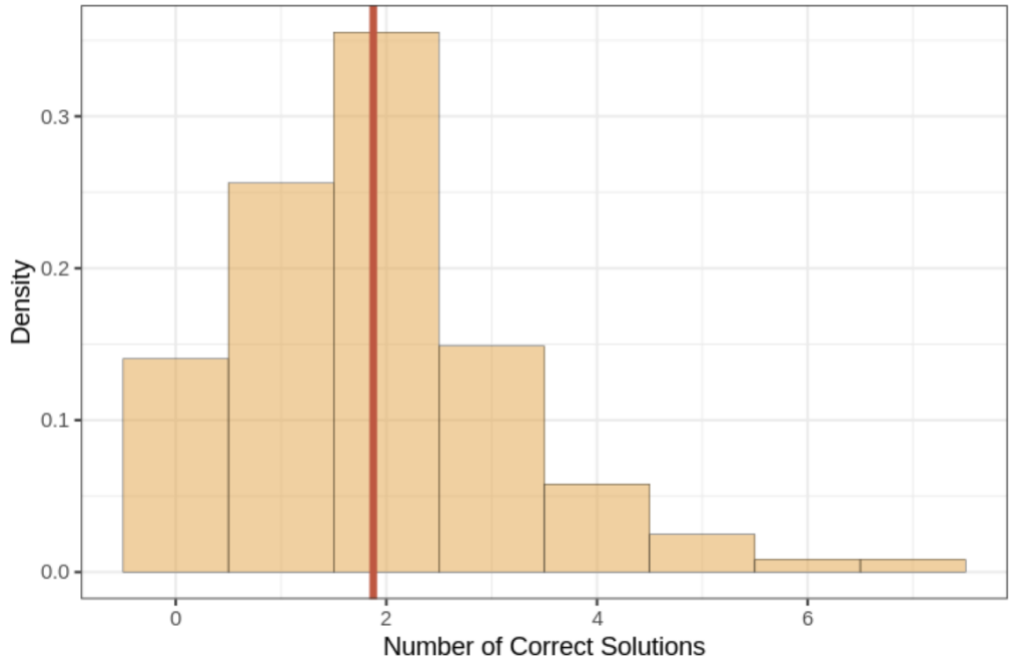

Figure 12. Histogram of number of correct solutions generated on the novel programming problem. The vertical red line represents the mean.

Descriptive statistics for scores on the learning measures for students in the Predict and Traditional conditions are shown in Table 9. Participants in the Predict condition scored higher on the learning assessment (Figure 13) (b_1=1.37, F(1, 119)=3.94, p = .05, *95% CI*[0.00, 2.75]) and generated more correct solutions to a novel programming task (Figure 14) (b_1=0.61, F(1, 119)=6.69, p = .011, *95% CI*[0.14, 1.08]) than participants in the Traditional condition. The effect of condition on score on the post lesson assessment remained statistically meaningful when gpa and need for cognition were included in the model (b_1=1.38, F(1, 116)=4.12, p = .05, *95% CI*[0.03, 2.72]). The effect of condition on the number of correct solutions also remained statistically meaningful when need for cognition and GPA were included in the model (b_1=1.38, F(1, 116)=4.12, p = .05, *95% CI*[0.03, 2.72]).

Table 9. Descriptive statistics and t-tests comparing post-survey measures of learning between the Predict and Traditional instructional conditions.

Time	Predict *M*(SD)	Traditional *M*(SD)	b_1 [95% CI]	*t*(df)	*p*
Score on learning assessment	9.90(3.72)	8.52(3.90)	1.38 [0.00,2.75]	1.99(118.91)	.05
Number of correct solutions	2.18(1.48)	1.57(1.09)	2.183 [0.14,1.08]	2.58(108.32)	.01

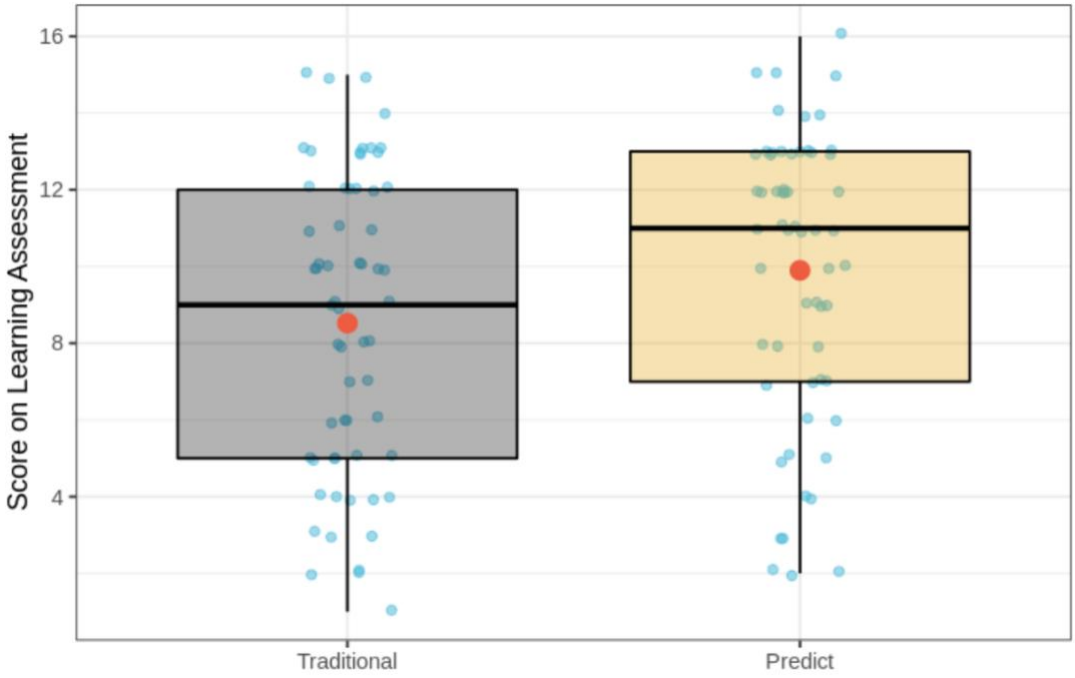

Figure 13. Boxplot of the effect of condition on score on the post-lesson assessment. Points represent the individual participants' scores. The horizontal line represents the median. The red dot represents the mean.

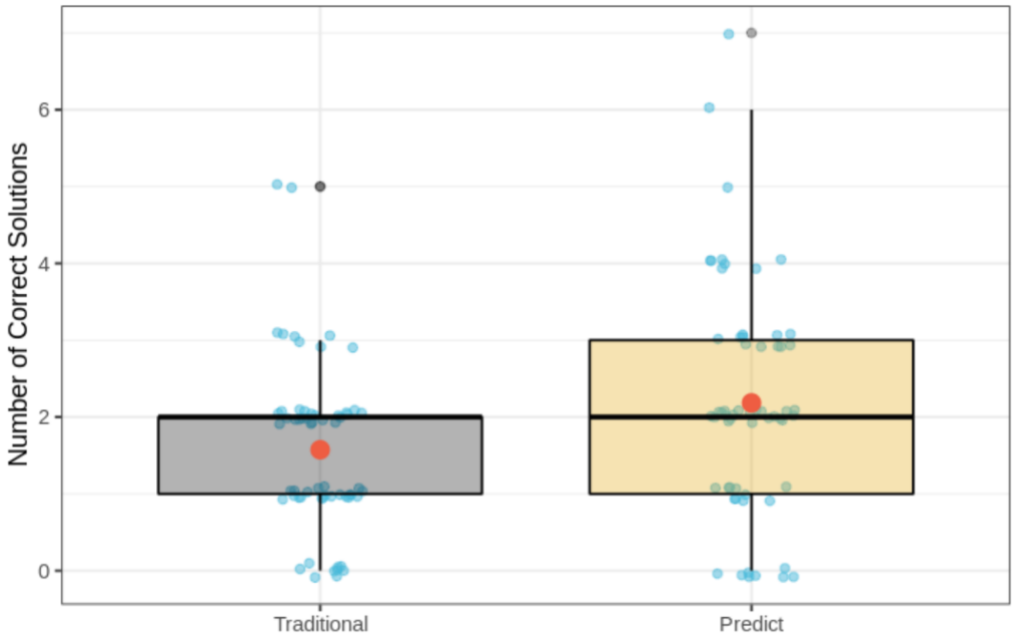

Figure 14. Boxplot of the effect of instructional condition on the number of correct solutions students generated on a novel programming problem. Points represent the individual participants' scores. The horizontal line represents the median. The red dot represents the mean.

Process Measures and their Relation to Learning, Motivation, and Emotion

Distributions of process measures, broken down by condition, and results from independent t-tests are shown in Table 10. Participants in Predict and Traditional conditions did not differ in word count (p = .06), duration (p = .06), or perceived intrinsic load (p = .37). However, compared to participants in the Traditional condition, participants in the Predict condition perceived the extraneous load to be lower (p = .005) and the germane cognitive load to be higher (p = .05).

Table 10. Descriptive statistics and t-tests comparing process measures between the Predict and Traditional instructional conditions.

	Predict *M*(SD)	Traditional *M*(SD)	b_1 [95% CI]	*t*(df)	*p*
Behavior					
Word count	74.88(43.47)	61.62(30.98)	13.26[-0.31,26.83]	-1.93 (106.53)	.06
Duration (s)	3560.20 (1492.26)	3073.52 (1327.60)	486.7[-21.55,994.91]	-1.89 (116.93)	.06
Performance	11.28(2.65)	14.18(3.20)	-2.90[-3.95,-1.84]	5.43 (115.69)	<.001
Cognitive load					

Intrinsic load	4.11(2.14)	4.45(2.03)	-0.34[-1.09,0.41]	0.90 (118.37)	.37
Extraneous load	2.38(2.04)	3.45(2.10)	-1.07[-1.81,-0.32]	2.83 (118.98)	.005
Germane load	7.23(1.95)	6.44(2.44)	0.79[-0.01,1.59]	-1.97 (114.17)	.05

Pairwise correlations between process measures are shown in Figure 15. Duration was correlated only with word count (r = 0.29), suggesting that students who wrote more words spent more time on the activity. Students who wrote more words also tended to estimate the extraneous load to be lower (r = -0.31) and estimated the germane cognitive load to be higher (r = 0.24). They also produced more correct responses on the activities during learning (r = 0.2). Similarly, students who produced more correct responses on the learning module activities estimated the extraneous load to be lower(r = -0.22) and the intrinsic load to be lower (r = -0.34).

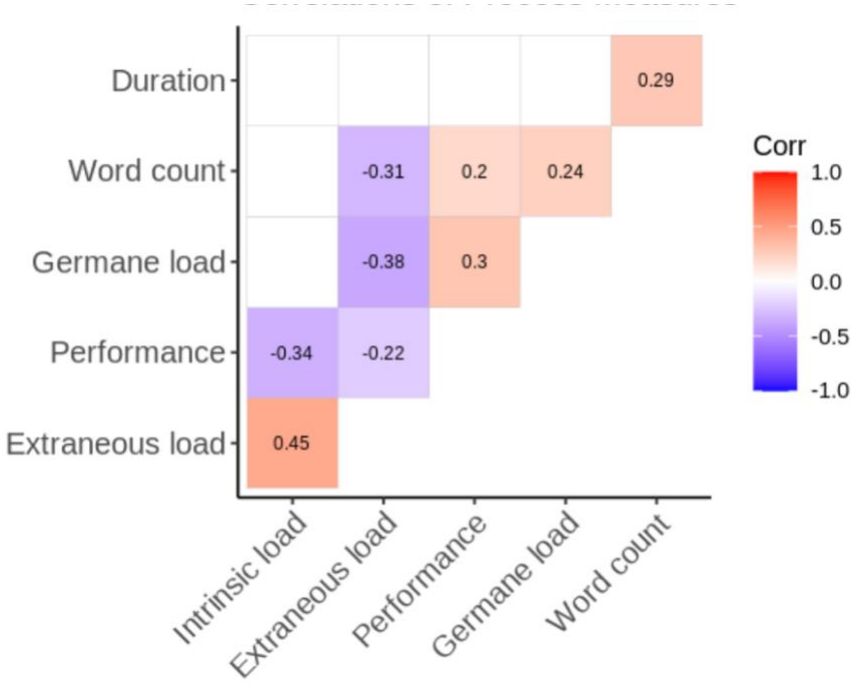

Figure 15. The pairwise correlation plot for performance variables. The numbers show the correlation coefficients. Warm colors indicate positive correlations. Cool colors indicate negative correlations. Non-significant correlations are blanked out.

Next, I investigated correlations between process measures (separated into categories: behavior and cognitive load). Pairwise correlations between process measures of behaviors and post-survey measures are shown in Figure 16.

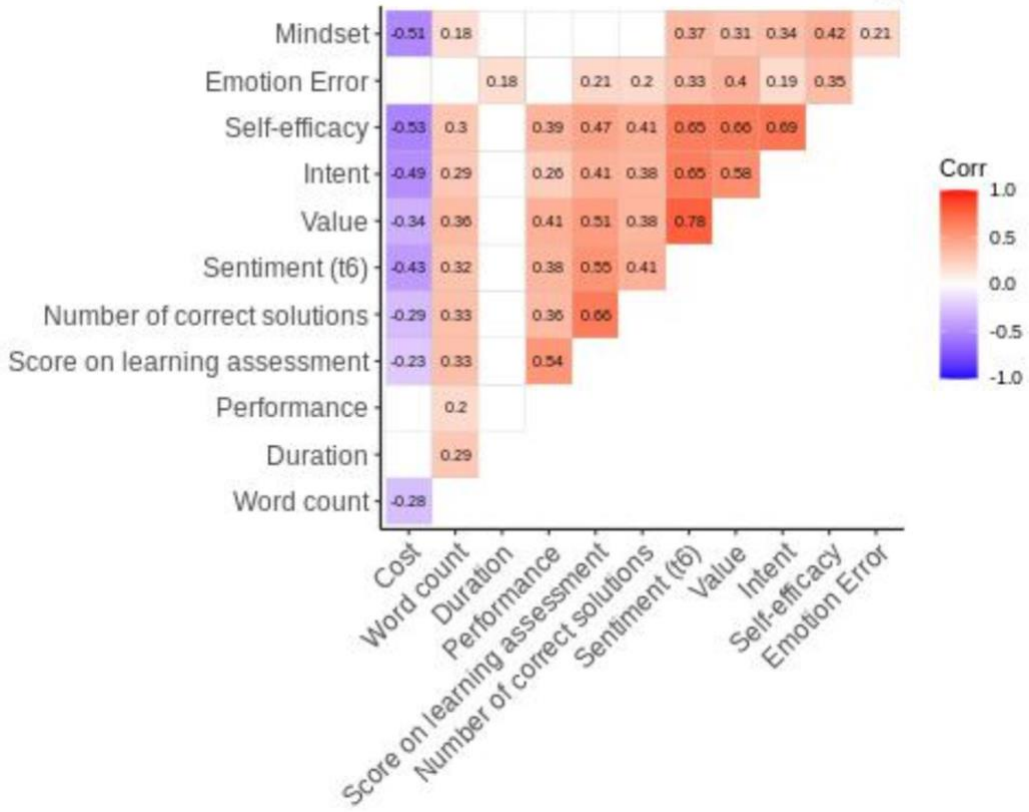

Figure 16. Pairwise correlations of process measures of behavior and post-survey measures. The numbers show the correlation coefficients. Warm colors indicate positive correlations. Cool colors indicate negative correlations. Non-significant correlations are blanked out.

Students who wrote more words in their summaries demonstrated increased self-efficacy (r = 0.3), more intent to take a programming course in the future (r = 0.29), greater interest in computer programming (r = 0.34), more positive sentiment on the post-survey (r =0.32), and more of a growth-oriented mindset towards computer programming. They also perceived the

value of learning programming to be greater (r = 0.36), scored higher on the post-lesson learning assessment (r = 0.33), and generated more correct solutions to the novel programming problem (r = 0.33). Performance across the 19 learning module activities was positively correlated with self-efficacy (r = 0.39), intent to take a programming course in the future (r = 0.26), programming interest (r = 0.38), perceived value of programming (r = 0.42), post-survey emotion (r = 0.38), as well as score on the learning assessment (r = 0.54), and number of correct solutions on the novel programming task (r = 0.36).

Notably, neither duration nor performance were correlated with perceived cost of learning programming while somewhat counterintuitively, higher word count was correlated with lower perceived cost. These findings suggest students' perceptions of cost are influenced by subjective appraisals rather than more objective measures, such as time or quantity of work. Further, duration (but not performance or word count) was weakly correlated with students' emotions in response to the hypothetical error scenario (r = 0.18), suggesting that students who spent more time overall, expressed more positive emotions at the end of the activity.

Next, I examined pairwise correlations between cognitive load and the post-survey measures (see Figure 17 for results).

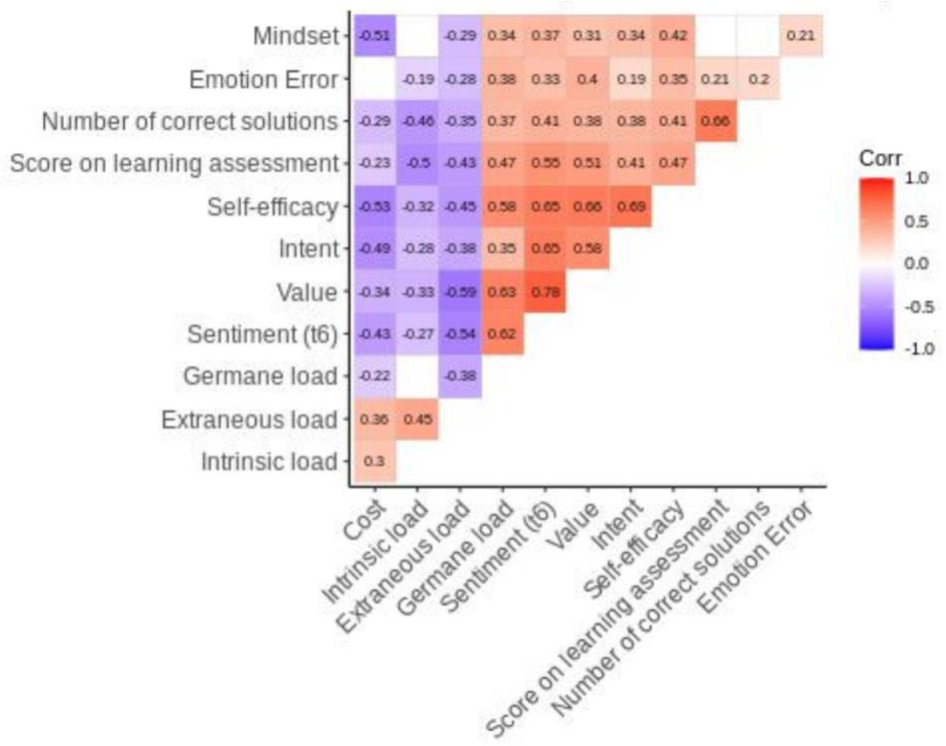

Figure 17. Pairwise correlations of process measures of estimated cognitive load and post-survey measures. The numbers show the correlation coefficients. Warm colors indicate positive correlations. Cool colors indicate negative correlations. Non-significant correlations are blank.

As expected, extraneous cognitive load was negatively correlated with all post-survey measures of learning, emotion, and motivation. Conversely, germane cognitive load was positively associated with all outcome measures of emotion, motivation, and learning. Intrinsic cognitive load was negatively correlated with all outcome measures except programming mindset.

The Role of Individual Differences

To explore whether the relationship between individual difference measures (GPA, attitudes towards computers, need for cognition, and initial motivation) and outcome measures (

emotion, motivation, and learning) varied as a function of condition, I visually compared the relationship between pre-survey and individual difference variables and student outcomes for students in the Predict and Traditional conditions.

Having a growth mindset towards programming positively predicted emotional response to error for participants in the Traditional condition, but it did not predict emotional response to error for participants in the Predict condition.

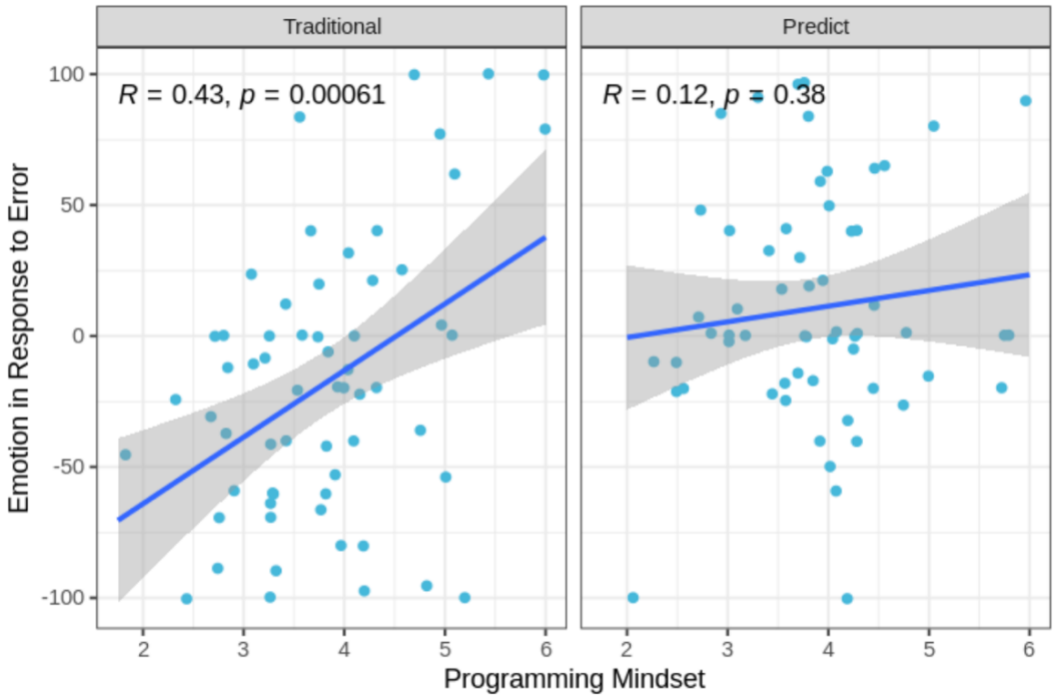

Figure 18. Faceted scatterplot comparing the relationship between programming mindset and emotion in response to error for participants in the Traditional and Predict conditions.

In addition, the strength of the relationship between attitudes towards computers and perceived cost of learning programming was weaker for participants in the Predict condition compared to participants in the Traditional condition.

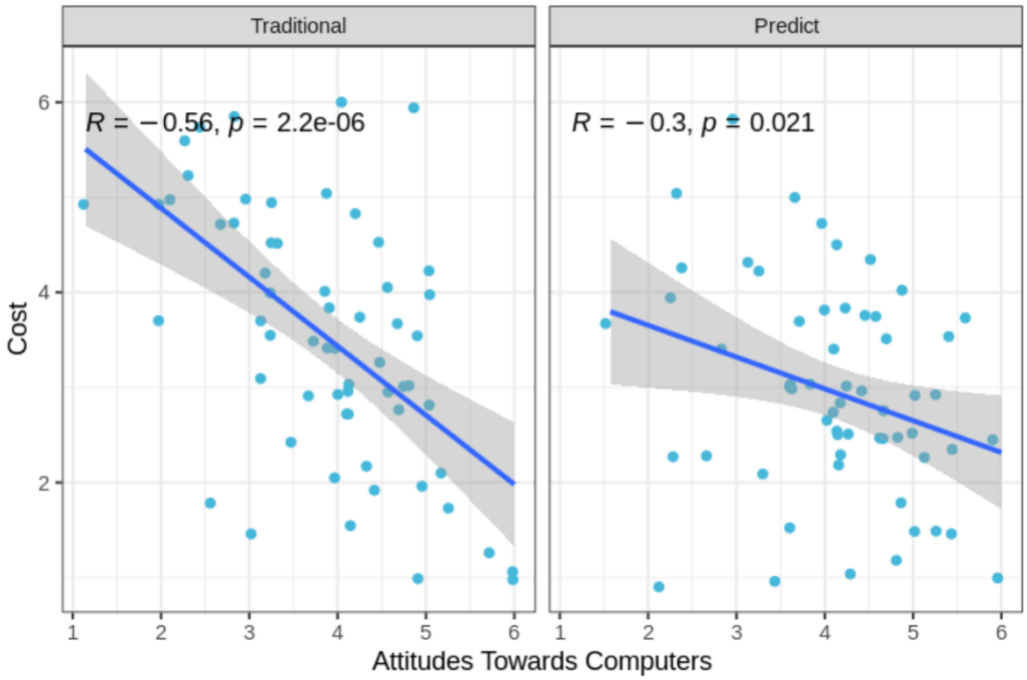

Figure 19. Faceted scatterplot comparing the relationship between computer attitudes and perceived cost of learning programming for participants in the Traditional and Predict conditions.

The relationship between individual difference measures and other measures of emotion, motivation, and learning did not visually appear to differ between students assigned to the Predict and Traditional condition. Taken together, these results suggest that in most cases, the effect of individual differences on emotion, motivation, and learning did not vary depending on the type of programming in the task. However, in some cases, predicting tasks may suppress the effect of programming mindset and attitudes towards computers.

Discussion

Students' struggles to learn programming are well documented in the literature. Prior studies indicate these struggles may be due in part to the tasks used to teach programming. In this

dissertation, I sought to develop a more comprehensive understanding of how programming tasks influence students' outcomes and experiences in introductory programming courses. Additionally, I sought to extend prior research investigating the effect of different programming tasks on instruction and explore the potential of predicting as an alternative strategy to teach programming.

Overall, I found that students randomly assigned to generate predictions demonstrated more positive outcomes than students randomly assigned to write and submit their own code. Specifically, students randomly assigned to generate predictions demonstrated more positive emotional trajectories, expressed increased motivation to learn programming, and performed better on measures of learning than students assigned to produce their own code. These findings complement prior studies that show benefits of predicting in complex learning environments.

Effect of predicting on emotions

Students assigned to the predicting condition demonstrated more positive emotional trajectories during learning. Though the two groups did not differ in their sentiment expressed at the start of the activity, students in the predict condition demonstrated more positive sentiment than students in the traditional condition as the lesson progressed. Importantly, the differences in the two groups first emerged as the task difficulty increased, suggesting that predicting may help buffer against negative emotions associated with challenges or setbacks. This hypothesis is further supported by the finding that students assigned to the predicting condition demonstrated more positive responses when shown a hypothetical programming error on the post-survey.

Effect of predicting on motivation

Students assigned to the predicting condition scored higher on some, but all measures of motivation. Compared to students assigned to write and submit code, students assigned to generate predictions perceived the cost of learning programming to be lower and demonstrated

greater intent to take a programming course in the future. However, students in the two groups did not differ in terms of other measures of motivation, including perceived value of programming. Condition was similarly not associated with motivational beliefs, including programming mindsets or programming self-efficacy.

When looking at students' open-ended responses about their beliefs about programming at the end of the activity, students in both conditions expressed a variety of views. For example, one student assigned to the Predict condition described programming as, "hard work. frustration. figuring out how to fix your code. breaking tables and cups because you can't figure out how to fix your code. enjoyment because you fixed your code. feeling of accomplishment because you finished a project. tedious. boring at times. problem solving." Another student assigned to the Predict condition described coding as, "Hard, long hours, complicated, no solutions."

One possible explanation for these findings is that programming tasks affected students' momentary appraisals but did not influence beliefs, which are harder to change. Because this experiment took place over just a few hours, it's possible that the time wasn't sufficient to socialize students' beliefs. An interesting direction for future research would be to investigate the effect of predicting versus other forms of instruction on students' beliefs when predicting is integrated into programming instruction over weeks or months rather than introduced in a few short activities.

Effect of Predicting on Learning

Students who generated predictions scored higher on the learning assessment and generated more correct solutions on a novel programming problem than students who wrote their own code, despite being presented with the same instructional content. One possibility is that generating predictions elicited emotional responses such as curiosity and interest, which facilitated knowledge acquisition and made students more motivated and open to generating

solutions on the post-lesson assessment. Students assigned to the Predict condition demonstrated more positive emotional trajectories during learning, providing initial evidence that predicting impacted students' subjective emotions and attitudes towards learning programming. Prior research has shown that emotions like curiosity can lead to greater exploration and facilitate making connections, which could lead to deeper and more connected understanding. Positive emotions during learning can also orient students' attention to feedback. Thus, students in the Predict condition may have learned more from the instruction than students in the Traditional condition.

Another possible explanation for this difference is that code writing activities increase the cognitive burden of learning programming. Students assigned to the predicting condition rated the extraneous cognitive load lower and the germane cognitive load higher than students assigned to the traditional condition, which suggests that some aspect of writing and submitting code increased the perceived cognitive load of learning from the perspective of beginning students. Students also perceived the cost of learning to be higher in the Traditional condition, even though the amount of time and effort (as evidenced by word count) did not meaningfully differ across the two conditions. One factor that may have increased the perceived cost of learning programming could be the extraneous cognitive load introduced by the added interactivity of the code writing activities. Another option could be the burden added by negative emotions during learning. Future research should investigate which aspects of code writing tasks might increase this perceived cognitive burden.

A third possible explanation is that predicting influenced cognitive processes known to benefit deeper learning. For instance, predicting could be seen as a form of retrieval, which is known to benefit encoding and memory (i.e., Karpicke & Roediger, 2008). Students in the predicting condition were asked to draw on what they learned to make predictions at multiple

timepoints during learning. It's possible that the effect of repeated testing alone facilitated deeper encoding and post-lesson performance. Predicting could also be considered a form of productive failure, which can prepare students for future learning (Kapur, 2008). When presented with ill-structured problems, students struggle and produce incorrect solutions, only to perform better on well-structured problems later compared to students presented with only well-structured problems (Kapur, 2008). One explanation is that ill-structured tasks cause students to engage in deeper cognitive processing such as drawing on prior knowledge to make up for limited information in the problem text. (McNamara, 2001). This cognitive processing prepares students to better process information and apply prior knowledge in subsequent structured tasks.

Another explanation is that well-structured tasks limit curiosity and exploration of the problem space (Reiser, 2004) and limit the types of strategies and approaches students use. If students are told to modify some code so that it runs correctly, they are limited to searching for explanations for why the code doesn't work. Instead, if students are asked to predict what the code will produce, they have to imagine multiple possible outcomes of what the code might do. The former would lead students to retrieve from memory knowledge of syntax and errors whereas the latter would lead students to elaborate on what they already know. When asked about obstacles they encountered during learning, students assigned to the Traditional condition tended to mention memorization. For example, one student said, "It got complicated, and I felt like I was being bombarded with too much new information to learn and regurgitate." Another student said, "it was hard to memorize some of the more complex codes." In the words of another student, "I found it a bit difficult to memorize all the rules and apply them all at once during the latter stage."

Finally, predicting could facilitate deeper learning by directing students' attention to relevant aspects of the problem. In this way, predicting could be seen as a form of comparing

contrasting cases (Loibl et al., 2020; Schwartz & Martin, 2004) that highlight the deeper structural aspects of the programming language. Contrasting cases were introduced by Schwartz and Bransford (1998) as a way to help students notice similarities and differences that highlight principles when introducing new concepts. The predicting activities in this study were designed to highlight subtle differences in lines of code and direct students' attention to features of programming syntax. For example, students were asked to predict what print("Hello, world") would produce and then compare the output to their prediction. It's possible the element of comparison could be driving the differences in learning outcomes between the two conditions.

Effect of Individual Differences Under Different Task Conditions

This study lacked statistical power to test interactions between individual difference measures and instructional conditions (predict vs traditional). However, visual inspection of the data suggests that predicting may influence the relationship between some aspects of students' initial motivation and their learning outcomes. For example, predicting seemed to suppress the effect of initial mindsets and attitudes towards computers on students' emotions in response to error and perceived cost of learning programming respectively.

One way that predicting might influence the effect of individual differences is by changing the goal structure of the learning environment. Prior research suggests that students' goals can be modified by situational cues about effort, evaluation, and the purpose of engaging in the learning task (Ames, 1999; Nicholls, 1984 for reviews). Students who have a growth mindset towards programming tend to adopt more mastery-oriented goals and approaches to learning. If mastery goals are cued by elements of the learning environment in the Predict condition, then we might expect the effect of programming mindsets to be less pronounced for students in the Predict but not Traditional condition. These cues would lead students in the Predict condition,

regardless of initial mindset or orientation, to adopt more adaptive goals and approaches toward learning, buffering against the effect of initial programming mindset.

Predicting might similarly buffer against the effect of negative attitudes towards computer programming by the implicit cues evoked during learning. Most programming instruction involves code writing activities similar to those in the Traditional condition. It is possible that the Traditional tasks evoked prior knowledge or beliefs about computer programming because they matched students' pre-existing schema about what programming requires. The Predict activities on the other hand, may have provided conflicting cues and thus not activated students' attitudes towards programming to the same extent of the tasks in the Traditional condition. Anecdotal evidence from students' open-ended responses suggests that some students in the Predict condition may even have been pleasantly surprised that the activity did not match their negative perceptions of computer programming. For example, one student stated, "I have found the format of the learning style very interesting and easy-to-follow. Initially, I thought it would be somewhat difficult because I have always seen coding as complex. However, this has made me understand how quickly I can be able to understand R coding and made me want to consider taking future courses."

Limitations and Future Directions

This study provides initial evidence of the potential of using predicting as a strategy to improve learning among students in the early stages of learning programming. However, there are a number of limitations that should be considered when interpreting the results.

Participants and Measures

First, participants were recruited from the psychology department at a competitive university, and thus may not be representative of the broader population of beginning programming students. Within the population, most students were high performing, as evidenced

by the high GPAs. Second, this study only investigated the effect of predicting versus traditional instruction using the R programming language. Prior research indicates that some programming languages may be easier to learn than others. Because we only looked at one programming language, it's not clear whether or not predicting might similarly benefit learners in the early stages of learning another, more complex language.

Another important limitation of this study is the measurement of student outcomes. Measures of emotion, motivation, and perceived cognitive load relied on self-report ratings, which may be less accurate than more objective behavioral measures, such as, for example, students' actual rather than anticipated future programming course engagement or more objective measures of cognitive load. In addition, sentiment ratings were captured at multiple timepoints throughout the activity. However, asking students to stop and rate their sentiment during learning can disrupt learning processes and impact students' experiences during learning. Less intrusive and more direct measures of emotions such as physiological measures or automated affect detection present a promising alternative to self-reported ratings and would be valuable to include in future research. Additionally, because measures of learning included problems that were relatively similar to those used during instruction, it is not clear whether predicting benefits 'far transfer' – performance on tasks that share fewer surface-level similarities with the content covered in the lesson.

Study Context

Although this study provides initial evidence that predicting may influence learning in a controlled experimental setting, it does not provide information on whether these findings extend to naturalistic learning contexts. Mastering computer programming takes years. And most people encounter programming, not as a single lesson, but in the context of a broader course or

series of courses. It will be important to study the effects of predicting in these more realistic contexts.

One related question is how the effect of predicting might vary based on the broader academic culture in which instruction takes place. For example, would predicting benefit learners the same way in a high-pressure environment in which passing the course is required for students to gain acceptance into a major as in low-stakes settings such as an elective? Another important question is whether the benefits of predicting can be observed over a longer period of time. For example, it would be interesting to see whether varying programming instruction in the first few weeks of a course would benefit students as the course progresses over weeks or even months.

Beyond looking at the effect of predicting over a longer period of time, it would be interesting to investigate whether predicting activities really do prepare students to learn more advanced programming concepts in the future. To investigate the effects of predicting on future learning and transfer, experiments could adopt a procedure similar to Schwartz & Bransford (1998) in which participants are provided initial instruction using either predicting or traditional methods, then provided the same instruction in a new concept. Such an approach would allow researchers to compare students' learning from the second programming lesson to answer questions about whether predicting prepares students to learn more deeply from subsequent instruction.

Another important question is how generating predictions might affect outcomes at different stages during learning. Research from cognitive psychology suggests that worked examples only benefit novice learners at the early stages of learning. Is predicting only beneficial for beginning students or could it also benefit students with more expertise? A second question concerns how predicting fits in with other learning strategies. In this experiment, we used only one instructional strategy in each condition. However, in a real classroom setting teachers use

multiple strategies and approaches together to enhance students' learning. How might interleaving predicting with other instructional activities influence student outcomes? Could introducing predicting activities alongside traditional instruction improve student outcomes?

Mechanisms Through which Predicting Affects Student Outcomes

Although the information in this study does not explain how predicting influences emotions, motivation, and learning, it does provide a basis for future studies of these mechanisms. Several questions could be used to investigate the potential mechanisms through which predicting influences student outcomes when learning computer programming.

One question is whether changing the nature of the predicting activity produces similar improvements to students' emotions, motivation , and learning. In this study we carefully designed predicting activities to highlight specific concepts. Would students experience similar benefits if asked to make a more general prediction? We are currently conducting a follow-up study to investigate this question. In this follow up study, students are randomly assigned to one of three instructional conditions: a Traditional condition similar to the Traditional condition in the present study, a Predict condition similar to the predict condition in this study where students make specific predictions by selecting from multiple choice options, and an open-ended Predict condition in which students generate their own predictions using an open-ended response option. Our hypothesis is that students in both the general and multiple choice Predict conditions will demonstrate more positive emotions and motivation than students in the Traditional condition, but students the multiple choice Predict condition will learn more than students in the Traditional and general Predict conditions because the multiple choice Predict task helps to direct students' attention to deeper features in the problem that will benefit learning and transfer.

Another follow-up question is how predicting compares to other well-researched strategies whose mechanisms for affecting learning are known. Comparing predicting to

activities like retrieval, or self-explanation and worked examples, might clarify the mechanisms through which predicting influences student outcomes. Similarly, investigating the affective outcomes of previously researched learning strategies would add to our knowledge of how these strategies work to improve student outcomes. Though many studies have investigated the cognitive benefits of strategies like retrieval, worked examples, and contrasting cases, almost none have investigated the effect of these strategies on students' emotional and motivational outcomes.

Finally, there were important features of the code-writing tasks that were not controlled for in this study that may have contributed to students' difficulties. Specifically, the Traditional code writing tasks included additional elements of interactivity and more complicated task instructions which could have introduced extra cognitive load or influenced students' emotions during learning. Students' open-ended responses to the obstacles encountered during learning provide some insight into this issue. For example, some students mentioned the slow loading time of the coding environment (i.e., "the computer was a little slow to execute the functions"; "the loading time for the coding to work"). Other students indicated they lacked the prior knowledge required to interpret code-writing instructions and error messages in their code.

Task instructions and feedback are plausible avenues through which code writing tasks might influence student outcomes. For instance, instructions in the Traditional condition varied with each new code-writing activity depending on the goal of the task. By contrast, instructions in the Predict condition remained relatively constant, as each task required them to do the same thing: predict what code would do. Keeping the instructions consistent may have reduced extraneous cognitive load, freeing up cognitive resources for students to focus on understanding what the code does. Since code writing is still an important part of learning programming, more research is needed to identify which elements of the code writing activities may have contributed

to students' poorer outcomes, and, how to design code-writing tasks in a way that reduces the impact of these features. Though these questions were not addressed in the present study, they would be a logical next step for future research.

Conclusion

In this randomized experiment comparing two instructional strategies to teach computer programming, evidence was found that generating predictions can lead beginning students to have more positive subjective emotional experiences, increased motivation, and more positive learning outcomes than modifying or writing code. This study is the first to our knowledge to test a theoretically informed hypothesis about the effect of different learning strategies for teaching programming on both students' cognitive and affective outcomes. Given the importance of cognitive and affective processes when learning computer programming, examining how different instructional approaches influence these processes in tandem is important to consider when designing programming instruction. This study also raises questions about the usefulness of popular instructional strategies for teaching programming especially for novice programmers in the early stages of learning. Though writing code may seem like the obvious way to introduce students to the practice of computer programming, asking students to write and submit their own code too early may exacerbate some of the challenges of learning programming.

Looking beyond computer programming, the findings from this study provide a glimpse of how learning tasks may impact multiple learning processes and underscore the need to consider cognition and affect together in educational research and design.

References

Altadmri, A., & Brown, N. C. C. (2015). 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 522-527. https://doi.org/10.1145/2676723.2677258

Ames, C. (1995). Achievement goals, motivational climate, and motivational processes. In *Motivation in sport and exercise* (pp. 161–176). Human Kinetics Books.

Ames, C., & Archer, J. (1987). Mothers' beliefs about the role of ability and effort in school learning. *Journal of Educational Psychology*, *79*(4), 409–414. https://doi.org/10.1037/0022-0663.79.4.409

Ames, C., & Archer, J. (1988). Achievement goals in the classroom: Students' learning strategies and motivation processes. *Journal of Educational Psychology*, *80*(3), 260–267. https://doi.org/10.1037/0022-0663.80.3.260

Bandura, A. (1977). Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review*, *84*(2), 191–215. https://doi.org/10.1037/0033-295X.84.2.191

Bandura, A. (1986). Fearful expectations and avoidant actions as coeffects of perceived self-inefficacy. *American Psychologist, 41*(12), 1389–1391. https://doi.org/10.1037/0003-066X.41.12.1389

Bandura, A., Barbaranelli, C., Caprara, G. V., & Pastorelli, C. (2001). Self-Efficacy Beliefs as Shapers of Children's Aspirations and Career Trajectories. *Child Development*, *72*(1), 187–206. https://doi.org/10.1111/1467-8624.00273

Barron, K., & Hulleman, C. (2014). Expectancy-Value-Cost Model of Motivation. In *International Encyclopedia of the Social & Behavioral Sciences* (pp. 503-509 (Vol. 8)). https://doi.org/10.1016/B978-0-08-097086-8.26099-6

Bayman, P., & Mayer, R. E. (1983). A diagnosis of beginning programmers' misconceptions of BASIC programming statements. *Communications of the ACM*, *26*(9), 677–679. https://doi.org/10.1145/358172.358408

Blanton, W. E., Wood, K. D., & Moorman, G. B. (1990). The Role of Purpose in Reading Instruction. *The Reading Teacher*, *43*(7), 486–493.

Bosch, N., Chen, Y., & D'Mello, S. (2014). It's Written on Your Face: Detecting Affective States from Facial Expressions while Learning Computer Programming. In S. Trausan-Matu, K. E. Boyer, M. Crosby, & K. Panourgia (Eds.), *Intelligent Tutoring Systems* (pp. 39–44). Springer International Publishing. https://doi.org/10.1007/978-3-319-07221-0_5

Bosch, N. & D'Mello, S. (2013). Programming with Your Heart on Your Sleeve: Analyzing the Affective States of Computer Programming Students. In H. C. Lane, K. Yacef, J. Mostow, & P. Pavlik (Eds.), *Artificial Intelligence in Education* (pp. 908–911). Springer. https://doi.org/10.1007/978-3-642-39112-5_143

Bosch, N. & D'Mello, S. (2017). The Affective Experience of Novice Computer Programmers. *International Journal of Artificial Intelligence in Education*, *27*(1), 181–206. https://doi.org/10.1007/s40593-015-0069-5

Bosch, N., D'Mello, S., & Mills, C. (2013). What Emotions Do Novices Experience during Their First Computer Programming Learning Session? In H. C. Lane, K. Yacef, J. Mostow, & P. Pavlik (Eds.), *Artificial Intelligence in Education* (pp. 11–20). Springer. https://doi.org/10.1007/978-3-642-39112-5_2

Braune, G., & Mühling, A. (2020). Learning to program: The gap between school world and everyday world. *Proceedings of the 15th Workshop on Primary and Secondary Computing Education*, 1–9. https://doi.org/10.1145/3421590.3421597

Brod, G. (2021). Predicting as a learning strategy. *Psychonomic Bulletin & Review*, *28*(6), 1839–1847. https://doi.org/10.3758/s13423-021-01904-1

Brod, G., Hasselhorn, M., & Bunge, S. A. (2018). When generating a prediction boosts learning: The element of surprise. *Learning and Instruction*, *55*, 22–31. https://doi.org/10.1016/j.learninstruc.2018.01.013

Cacioppo, J. T., Petty, R. E., & Feng Kao, C. (1984). The Efficient Assessment of Need for Cognition. *Journal of Personality Assessment*, *48*(3), 306–307. https://doi.org/10.1207/s15327752jpa4803_13

Cañas, J. J., Bajo, M. T., & Gonzalvo, P. (1994). Mental models and computer programming. *International Journal of Human-Computer Studies*, *40*(5), 795–811. https://doi.org/10.1006/ijhc.1994.1038

Clancy, M. J., & Linn, M. C. (1999). Patterns and pedagogy. *ACM SIGCSE Bulletin*, *31*(1), 37–42. https://doi.org/10.1145/384266.299673

Clore, G. L., Gasper, K., & Garvin, E. (2001). Affect as information. In J. P. Forgas (Ed.), *Handbook of affect and social cognition* (pp. 121–144). Lawrence Erlbaum Associates Publishers.

DataCamp Light (2020). Github repository. https://github.com/datacamp/datacamp-light

Davies, S. P. (1993a). Models and theories of programming strategy. *International Journal of Man-Machine Studies*, *39*(2), 237–267. https://doi.org/10.1006/imms.1993.1061

Davies, S. P. (1993b). The structure and content of programming knowledge: Disentangling training and language effects in theories of skill development. *International Journal of Human–Computer Interaction*, *5*(4), 325–346. https://doi.org/10.1080/10447319309526072

D'Mello, S. & Graesser, A. (2011). The half-life of cognitive-affective states during complex learning. *Cognition and Emotion*, *25*(7), 1299–1308. https://doi.org/10.1080/02699931.2011.613668.

Dweck, C. (1999). Mindset: The New Psychology of Success (Ballentine, New York, 2006). Self-theories: Their Role in Motivation, Personality, and Development.

Elliot, A. J. (1999). Approach and avoidance motivation and achievement goals. *Educational Psychologist*, *34*(3), 169–189. https://doi.org/10.1207/s15326985ep3403_3

Ellis, H. C., & Ashbrook, P. W. (1989). The "state" of mood and memory research: A selective review. *Journal of Social Behavior & Personality*, *4*(2), 1–21.

Ellis, H. C., Seibert, P. S., & Varner, L. J. (1995). Emotion and Memory: Effects of Mood States on Immediate and Unexpected Delayed Recall. *Journal of Social Behavior and Personality*, *10*(2), 349–362.

Gaspard, H., Häfner, I., Parrisius, C., Trautwein, U., & Nagengast, B. (2017). Assessing task values in five subjects during secondary school: Measurement structure and mean level differences across grade level, gender, and academic subject. *Contemporary Educational Psychology*, *48*, 67–84. https://doi.org/10.1016/j.cedpsych.2016.09.003

Guilford, J. P. (1967). Creativity: Yesterday, Today and Tomorrow. *The Journal of Creative Behavior*, *1*(1), 3–14. https://doi.org/10.1002/j.2162-6057.1967.tb00002.x

Herold, D. M., Davis, W., Fedor, D. B., & Parsons, C. K. (2002). Dispositional Influences on Transfer of Learning in Multistage Training Programs. *Personnel Psychology*, *55*(4), 851–869. https://doi.org/10.1111/j.1744-6570.2002.tb00132.x

Hill, T., Smith, N. D., & Mann, M. F. (1987). Role of efficacy expectations in predicting the decision to use advanced technologies: The case of computers. *Journal of Applied Psychology*, *72*(2), 307–313. https://doi.org/10.1037/0021-9010.72.2.307

Hsu, C.-Y., Tsai, C.-C., & Liang, J.-C. (2011). Facilitating Preschoolers' Scientific Knowledge Construction via Computer Games Regarding Light and Shadow: The Effect of the Prediction-Observation-Explanation (POE) Strategy. *Journal of Science Education and Technology*, *20*(5), 482–493. https://doi.org/10.1007/s10956-011-9298-z

Isen, A. M., (2000). Some Perspectives on Positive Affect and Self-Regulation. *Psychological Inquiry*, *11*(3), 184–187.

Jiang, Y., Rosenzweig, E. Q., & Gaspard, H. (2018). An expectancy-value-cost approach in predicting adolescent students' academic motivation and achievement. *Contemporary Educational Psychology*, *54*, 139–152. https://doi.org/10.1016/j.cedpsych.2018.06.005

Kapur, M. (2008). Productive Failure. *Cognition and Instruction*, *26*(3), 379–424. https://doi.org/10.1080/07370000802212669

Karpicke, J. D., & Roediger, H. L. (2008). The Critical Importance of Retrieval for Learning. *Science*, *319*(5865), 966–968.

Knuth, D. E. (1974). *Computer Programming as an Art*. *17*(12), 7.

Kosovich, J. J., Hulleman, C. S., Barron, K. E., & Getty, S. (2015). A practical measure of student motivation: Establishing validity evidence for the Expectancy-Value-Cost Scale in middle school. *The Journal of Early Adolescence*, *35*(5-6), 790–816. https://doi.org/10.1177/0272431614556890.

Kwon, O. N., Park, J. H., & Park, J. S. (2006). Cultivating divergent thinking in mathematics through an open-ended approach. *Asia Pacific Education Review*, *7*(1), 51–61. https://doi.org/10.1007/BF03036784

Le, N.-T., Boyer, K. E., Chaudry, B., Di Eugenio, B., Hsiao, S. I.-H., & Sudol-DeLyser, L. A. (2013). The First Workshop on AI-supported Education for Computer Science (AIEDCS). In H. C. Lane, K. Yacef, J. Mostow, & P. Pavlik (Eds.), *Artificial Intelligence in Education*

(Vol. 7926, pp. 947–948). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-39112-5_159

Liang, J-C., Su, Y-C., Tsai, C-C. (2015). The assessment of Taiwanese college students' conceptions of and approaches to learning computer science and their relationships. *The Asia-Pacific Education Researcher, 24*(4). 557-567. https://doi.org/s40299-014-0201-6

Liew, C., & Treagust, D. (1995). A Predict-Observe-Explain Teaching Sequence for Learning about Students' Understanding of Heat and Expansion Liquids. *Australian Science Teachers Journal*, *41*.

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

Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *ACM SIGCSE Bulletin*, *41*(3), 161–165. https://doi.org/10.1145/1595496.1562930

Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin*, *38*(3), 118–122. https://doi.org/10.1145/1140123.1140157

Loibl, K., Tillema, M., Rummel, N., & van Gog, T. (2020). The effect of contrasting cases during problem solving prior to and after instruction. *Instructional Science*, *48*(2), 115–136. https://doi.org/10.1007/s11251-020-09504-7

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. *Proceedings of the Fourth International*

Workshop on Computing Education Research, 101–112. https://doi.org/10.1145/1404520.1404531

Luxton-Reilly, A., Sheard, J., Szabo, C., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., & Scott, M. J. (2018). Introductory programming: A systematic literature review. *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2018 Companion*, 55–106. https://doi.org/10.1145/3293881.3295779

Ma, L. (2008). *Investigating and Improving Novice Programmers' Mental Models of Programming Concepts*.

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2011). Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, *21*(1), 57–80. https://doi.org/10.1080/08993408.2011.554722

Maricuţoiu, L. (2006). Emotional response to computer error messages. *Psihologia Resurselor Umane*

Martocchio, J. J., & Dulebohn, J. (1994). Performance Feedback Effects in Training: The Role of Perceived Controllability. *Personnel Psychology*, *47*(2), 357–373. https://doi.org/10.1111/j.1744-6570.1994.tb01729.x

Martocchio, J. J., & Judge, T. A. (1997). Relationship between conscientiousness and learning in employee training: Mediating influences of self-deception and self-efficacy. *Journal of Applied Psychology*, *82*(5), 764–773. https://doi.org/10.1037/0021-9010.82.5.764

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *Working Group Reports from*

ITiCSE on Innovation and Technology in Computer Science Education, 125–180. https://doi.org/10.1145/572133.572137

McNamara, D. S. (2001). Reading both high-coherence and low-coherence texts: Effects of text sequence and prior knowledge. *Canadian Journal of Experimental Psychology/Revue Canadienne de Psychologie Expérimentale*, *55*(1), 51–62. https://doi.org/10.1037/h0087352

McNamara, D. S., & Scott, J. L. (2001). Working memory capacity and strategy use. *Memory & Cognition*, *29*(1), 10–17. https://doi.org/10.3758/BF03195736

Medeiros, R. P., Ramalho, G. L., & Falcão, T. P. (2019). A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. *IEEE Transactions on Education*, *62*(2), 77–90. https://doi.org/10.1109/TE.2018.2864133

Meinhardt, J., & Pekrun, R. (2003). Attentional resource allocation to emotional events: An ERP study. *Cognition and Emotion*, *17*(3), 477–500. https://doi.org/10.1080/02699930244000039

Miller, K., Lasry, N., Chu, K., & Mazur, E. (2013). Role of physics lecture demonstrations in conceptual learning. *Physical Review Special Topics - Physics Education Research*, *9*(2), 020113. https://doi.org/10.1103/PhysRevSTPER.9.020113

Morrison, B. B., Dorn, B., & Guzdial, M. (2014). Measuring cognitive load in introductory CS: Adaptation of an instrument. *Proceedings of the Tenth Annual Conference on International Computing Education Research*, 131–138. https://doi.org/10.1145/2632320.2632348

Munezero, M., Montero, C. S., Sutinen, E., & Pajunen, J. (2014). Are They Different? Affect, Feeling, Emotion, Sentiment, and Opinion Detection in Text. *IEEE Transactions on Affective Computing*, *5*(2), 101–111. https://doi.org/10.1109/TAFFC.2014.2317187.

NAEP - 2014 Technology and Engineering Literacy. (n.d.). NAEP - 2014 Technology and Engineering Literacy. Retrieved July 5, 2022, from https://www.nationsreportcard.gov/tel_2014/#results/overall

Nicholls, J. (1984). Achievement motivation: Conceptions of ability, subjective experience, task choice, and performance. *Psychological Review, 91,* 328-346.

Nichols, J., & Miller, A. T. (1984). The development of Achievement-related judgment processes. Advances in Motivation and Achievement: the development of Achievement Motivation, 185-218.

Nolan, T. E. (1991). Self-Questioning and Prediction: Combining Metacognitive Strategies. *Journal of Reading*, *35*(2), 132–138.

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of Learning in Novice Programmers. *Journal of Educational Computing Research*, *2*(1), 37–55. https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL

Perkins, D. N. & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, 213–229.

Philpott, A., Robbins, P., & Whalley, J. L. (n.d.). *Assessing the Steps on the Road to Relational Thinking*.

Qian, Y. & Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Transactions on Computing Education*, *18*(1), 1:1-1:24. https://doi.org/10.1145/3077618

Ramalingam, V. & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy and group analyses of novice programmer self-efficacy.

Journal of Educational Computing Research, 19(4). https://doi.org/10.2190/C670-Y3C8-LTJ1-CT3P

R Core Team (2019). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/.

Reiser, B. J. (2004). Scaffolding Complex Learning: The Mechanisms of Structuring and Problematizing Student Work. In *The Journal of the Learning Sciences*. Psychology Press. https://doi.org/10.1207/s15327809jls1303_2

Renumol, V. G., Janakiram, D., & Jayaprakash, S. (2010). Identification of Cognitive Processes of Effective and Ineffective Students During Computer Programming. *ACM Transactions on Computing Education*, *10*(3), 10:1-10:21. https://doi.org/10.1145/1821996.1821998

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

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

Sajaniemi, J. & Navarro-Prieto, R. (2005). *Roles of variables in experts' programming knowledge*.

Salguero, A., Griswold, W. G., Alvarado, C., & Porter, L. (2021). Understanding Sources of Student Struggle in Early Computer Science Courses. *Proceedings of the 17th ACM Conference on International Computing Education Research*, 319–333. https://doi.org/10.1145/3446871.3469755

Schnotz, W., & Kürschner, C. (2007). A Reconsideration of Cognitive Load Theory. *Educational Psychology Review*, *19*(4), 469–508. https://doi.org/10.1007/s10648-007-9053-4

Schunk, D. H. (2012). Social cognitive theory. In *APA educational psychology handbook, Vol 1: Theories, constructs, and critical issues* (pp. 101–123). American Psychological Association. https://doi.org/10.1037/13273-005

Schwartz, D. L., & Bransford, J. D. (1998). A Time for Telling. *Cognition and Instruction*, *16*(4), 475–5223. https://doi.org/10.1207/s1532690xci1604_4

Schwartz, D. L., Chase, C. C., Oppezzo, M. A., & Chin, D. B. (2011). Practicing versus inventing with contrasting cases: The effects of telling first on learning and transfer. *Journal of Educational Psychology*, *103*(4), 759–775. https://doi.org/10.1037/a0025140

Schwartz, D. L., & Martin, T. (2004). Inventing to Prepare for Future Learning: The Hidden Efficiency of Encouraging Original Student Production in Statistics Instruction. *Cognition and Instruction*, *22*(2), 129–184. https://doi.org/10.1207/s1532690xci2202_1

Schwarz, N., & Clore, G. L. (2003). Mood as Information: 20 Years Later. *Psychological Inquiry*, *14*(3–4), 296–303. https://doi.org/10.1080/1047840X.2003.9682896

Simon, B., & Hanks, B. (2008). First-year students' impressions of pair programming in CS1. *Journal on Educational Resources in Computing*, *7*(4), 5:1-5:28. https://doi.org/10.1145/1316450.1316455

Sirkiä, T., & Sorva, J. (2012). Exploring programming misconceptions: An analysis of student mistakes in visual program simulation exercises. *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, 19–28. https://doi.org/10.1145/2401796.2401799

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction*, *4*(4), 295–312. https://doi.org/10.1016/0959-4752(94)90003-5

Sweller, J. (2010). Element Interactivity and Intrinsic, Extraneous, and Germane Cognitive Load. *Educational Psychology Review*, *22*(2), 123–138. https://doi.org/10.1007/s10648-010-9128-5

Sweller, J., & Cooper, G. A. (1985). The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra. *Cognition and Instruction*, *2*(1), 59–89. https://doi.org/10.1207/s1532690xci0201_3

Traver, V.J.(2010). On compiler error messages: What they say and what they mean. *Advances in Human-Computer Interaction, 2010*. https://doi.org/10.1155/2010/602570

Umapathy, K., Ritzhaupt, A. D., & Xu, Z. (2020). College Students' Conceptions of Learning of and Approaches to Learning Computer Science. *Journal of Educational Computing Research*, *58*(3), 662–686. https://doi.org/10.1177/0735633119872659

Venables, A., Tan, G., & Lister, R. (2009). A closer look at tracing, explaining and code writing skills in the novice programmer. *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*, 117–128. https://doi.org/10.1145/1584322.1584336

Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, 39–44. https://doi.org/10.1145/2591708.2591749

Webster, J., & Martocchio, J. J. (1992). Microcomputer Playfulness: Development of a Measure with Workplace Implications. *MIS Quarterly*, *16*(2), 201–226. https://doi.org/10.2307/249576

Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Ajith Kumar, P. K., & Prasad, C. (2006). *An Australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO taxonomies*. https://opus.lib.uts.edu.au/handle/10453/5050.

White, R. & Gunstone, R. (1992). Prediction-observation-explanation. Probing understanding, 4, 44-64.

Wigfield, A. (1994). Expectancy-value theory of achievement motivation: A developmental perspective. *Educational Psychology Review*, *6*(1), 49–78. https://doi.org/10.1007/BF02209024.