The Story in the Code: Programmers Show Distinct Brain Responses to Form and Meaning Violations During Python Comprehension

Chu-Hsuan Kuo

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:

Chantel S. Prat, Chair

Scott Murray

Ariel Starr

Andrea Stocco

Program Authorized to Offer Degree:

Department of Psychology

University of Washington

Abstract

The Story in the Code: Programmers Show Distinct Brain Responses to Meaning and Form Violations During Python Comprehension

Chu-Hsuan Kuo

Chair of the Supervisory Committee:

Chantel S. Prat

Department of Psychology

Despite efforts to bolster computer programming education, programming remains a difficult skill to acquire that we do not fully understand. Using the event-related potential method, this dissertation investigates how programmers process meaning and form during code comprehension as a method for understanding their underlying mental models of programming knowledge. Programmers exhibited distinct brain responses to semantic violations (N400) and structural violations (P600), similar to other symbolic systems such as natural language and mathematics. Programming expertise was associated with more prominent P600 effects, indicating a stronger sensitivity to structural cues. This dissertation presents the first evidence that programmers process meaning and form as distinct sources of information that jointly influence code comprehension, and that the construction, retrieval, and application of schemas informs programming expertise.

Table of Contents

In recent decades, computer programming has changed from being a skill predominantly used by specialists to one that allows people from all parts of society to navigate an increasingly technological world. Education systems around the globe have responded to this increase in demand in different ways—from considering coding classes as a foreign language option in high schools to requiring coding-related coursework for students in K-12 (National Education Policy, 2020; Nichols, 2021; Seow et al, 2019). However, these efforts to bolster computer literacy are limited in efficacy, as they focus on access to programming classes rather than on the content or format of programming education. Without an improved understanding of why learning to program is difficult in the current educational climate, coding will remain inaccessible to the reported 28% to 50% of students worldwide who drop their programming classes (Bennedsen & Caspersen, 2007, 2019; Watson & Li, 2014). This dissertation attempts to address a gap in the literature by applying existing theories on the acquisition of symbolic systems (e.g., natural language and mathematics) to the study of programming language comprehension in learners of varying skill levels.

Over twenty years ago, Jenkins (2002) proposed that "the way in which programming is taught and learned is fundamentally broken" because educators do not fully understand "the difficulties and complexities faced by the students." In this dissertation, I propose that some of this complexity is shared by other symbolic systems, like natural language and mathematics, which have been more thoroughly researched. Specifically, each of these systems requires declarative knowledge that maps symbols to meaning, and a set of rules that combine symbols to manipulate this meaning (e.g., Fedorenko et al., 2019). For example, understanding a computer program, like reading a paragraph written in a natural language, requires parsing letters and digits into meaningful units (e.g., functions, keywords, and variables) which are further arranged

into chunks of code that perform specific manipulations (e.g., adding or printing variables). Depending on the complexity of the code, processing these chunks may be further facilitated by knowledge schemas (e.g., how to print text in a programming console) that contribute to understanding the overall goal of the program at the highest-level. To generate code, a programmer needs to engage in the reverse process of translating an abstract goal into a program, which requires assembling chunks of code in a manner that will convey their intentions accurately (Federenko et al., 2019).

Prior research indicates that expertise in a variety of domains is related to the flexible application of schemas, or common patterns and rules, that shape individuals' representations of a problem or task, otherwise known as mental models (Curtis, 1984; Dalbey & Linn, 1985; Johnson-Laird, 1983; Mayer, 1981; Rouse & Morris, 1986; Wilson & Rutherford, 1989). On the contrary, novices struggle because they do not have the tools to represent their goals or intended meaning with abstract symbols in accurate and efficient ways (Mayer, 1981; Qian & Lehman, 2017). This creates a heavier reliance on slower, bottom-up processing of individual units (Lipshitz & Shaul, 1997; Robins et al., 2013). If this is true for programming knowledge as well, then I argue that we cannot fully address inadequacies in programming education, nor understand individual differences in learning to program, until we gain a deeper appreciation for the mental models constructed by successful programmers. To do so, we must turn our attention to the features that computer programming shares with other symbolic systems—the processing of meaning and form—by focusing on the neural mechanisms that underpin code comprehension.

The goal of this dissertation is to investigate the nature of mental models used by programmers by examining the neural signatures of unit-by-unit (aka "online") code comprehension. By leveraging what is known about the relationship between complex skill

acquisition and mental model creation, I aim to gain a more complete understanding of how meaning and form are processed by programmers with varying coding proficiency and natural language experience.

1.1. The Role of Schemas in the Construction of Mental Models During Task Performance

When acquiring a new skill like programming, generating effective mental models of the goal of a snippet of code can facilitate learning and future use of that skill (Mayer, 1981). A mental model is a representation of domain-specific tasks that individuals manipulate to facilitate decision making or problem solving (Collins & Gentner, 1987; Gentner & Stevens, 2014; Greca & Moreira, 2000; Johnson-Laird, 1983; 2004; Rouse & Morris, 1986; Wilson & Rutherford, 1989). The efficacy of a mental model in guiding behavior depends on its ability to account for the unique information in a given context. However, research on complex skill acquisition has shown that experts do not build each mental model from the ground up (e.g., engaging in bottom-up processes alone); instead, they draw upon a series of memory templates, or schema, that are flexible enough to generalize across variations of the scenario (Carroll, 1984; Gentner & Stevens, 1983; Norman, 1983; Staggers & Norcio, 1993).

This dynamic interaction between bottom-up comprehension processes and the use of top-down contextual memory structures (or schemas) to create mental models allows an individual to understand and operate on unfamiliar problems efficiently (Wilson & Rutherford, 1989). For example, a programmer who is asked to write a program that calculates someone's taxes may build a mental model for the task that is based on a series of simpler schemas (e.g., how to multiply dollar amounts by some percentage). These same schemas may be applied to other tasks such as how to calculate the tip at a restaurant.

Where does knowledge that forms these basic building blocks, or schemas, come from? The success of a mental model often depends on the schemas that we form based on previous experiences. Schemas are pieces of knowledge in memory that can be flexibly applied to form mental models that guide behavior, facilitate encoding of new information, and expedite the retrieval of facts (Bransford, 1979; Ghosh & Gilboa, 2014). They are abstracted shortcuts that we create to simplify future encounters with similar objects or events, reducing the need to relearn appropriate responses to familiar stimuli.

During skill acquisition, schemas can be developed to facilitate the understanding and usage of common knowledge or actions in service of a larger goal (McVee et al., 2005). For example, as expertise develops, a programmer will form schemas that represent the most frequently used code structures that achieve simple actions (e.g., adding or subtracting numbers, printing text to a console). Rather than encoding these schemas purely as a series of operators and symbols, a programmer also attaches higher-level meaning to the collective structure, such as "calculation of a running sum." By forming pre-existing schemas, a programmer can retrieve the relevant ones from memory to flesh out their desired program (e.g., calculating taxes) without resorting to lower-level creation of the same code structures every time they are needed. After the correct schemas are retrieved, a programmer can organize them according to the program design flow that accomplishes their ultimate goal; this final step is the realization of the mental model for the task.

By constructing a mental model with the help of schemas, we can accomplish tasks by processing information in a top-down, predictive manner rather than in a unit-by-unit, bottom-up fashion. Even though mental models differ in their content across domains, they provide similar overarching benefits such as faster processing and the opening up of resources for future

learning. Above all, mental models allow for creative problem-solving instead of the mere parroting of rote-memorized facts; in other words, they are evidence of *understanding* in addition to *knowing* (Staggers & Norcio, 1993).

1.1.1. Examples of schemas influencing mental models in non-programming domains. Schemas influence the construction of mental models because they are created via one of the most powerful learning mechanisms: relating new content with past experiences (Barnett & Ceci, 2002; Gentner & Medina, 1998; Hofstadter, 2001). In doing so, schemas provide us with stability in the face of uncertainty (Staggers & Norcio, 1993). One of the most well-known examples of schemas influencing comprehension comes from a study by Chase and Simon (1973) on chess players; they found that chess experts could recreate chess positions with higher accuracy than chess novices, but only if the positions were related to a possible aspect of game play. If the chess positions were random or impossible configurations, chess experts performed similarly to chess novices. In other words, chess experts remembered the position of pieces by retrieving schemas related to feasible scenarios associated with game play.

Additional studies have replicated and extended this finding, indicating that it is not the experience alone that separated chess experts from chess novices, but also the depth of their knowledge (Lane & Chang, 2018). After playing countless matches, chess experts assemble many configurations of the same pieces on the board in organized manners. Eventually, they come to see the patterns of pieces as a situation in which players interact with each other to produce a larger game that can play out in various ways. The most proficient chess players can visualize multiple outcomes for each move they consider, which they accomplish by stitching together schemas of configurations they have previously experienced. For example, there are documented opening moves that serve different purposes depending on the preferred plan of

attack (CHESScom, 2022). As players obtain more chess knowledge, they develop schemas for openings, endings, and specific gameplay scenarios in between, allowing them to adjust to their opponent in deliberate rather than reactive ways. By engaging in predictive processing of game outcomes, such as imagining the subsequent moves that could happen as a result of advancing a pawn, experts rely on schemas of meaningful chess patterns to simulate mental models of the overall game (Gobet & Simon, 1998). On the other hand, novices, who lack the experience to form meaning-based schemas, often rely on rote memorization of the individual pieces to construct mental models of the situation at hand, which leads them to focus on individual turns rather than planning out steps (Milojkovic, 1982). This, in turn, limits their ability to generate mental models of an entire game.

The relation between expertise and well-organized schemas was subsequently demonstrated across many domains including mathematics (Cooper & Sweller, 1987; Owen & Sweller, 1985), music (Cannone & Aucouturier, 2016; Krumhansl & Castellano, 1983; Leman, 2012; Shevy, 2008; Steedman, 1996; Zatorre & Krumhansl, 2002), and physics (Greca & Moreira, 2000; Roschelle & Greeno, 1987; Rouse & Morris, 1986). Perhaps the most universally relatable example of the benefits of schemas in the construction of mental models is in natural language processing, which is often developed from birth. During the initial phases of natural language acquisition, a learner engages in bottom-up processing of every word individually in order to extract meaning from sentences (Hoff, 2013). As an individual becomes more proficient in a language, they move toward top-down processing by combining advanced lexico-semantics and syntax with their schemas of world knowledge to generate complete mental models of the larger text or conversation at hand (Hoff, 2013; McCarthy, 1946). The more schemas we own about common scenarios, actions, and objects, the more they assist in language comprehension

post-acquisition. For example, Bransford and Johnson's (1972) seminal work showed that readers who were provided with the context for a paragraph containing vague actions ("washing clothes") performed better on a recall task than readers who were only provided with the paragraph. The presence of context allowed readers to encode the sentences as meaningful steps in a familiar process, whereas readers who did not have context had to resort to pure memorization of words without a clear guide to put them together. In a follow-up study, Bransford and Johnson (1972) showed that providing readers with the context after they read the paragraph did not boost recall, indicating that schemas are beneficial only when they can support the creation of mental models at the beginning of a task. Since then, numerous studies have documented the facilitating effect of schemas on mental models for text and discourse in native and second language comprehension (Anderson, 1977; Barnitz, 1986; Carrell, 1984).

Through interacting with the world, we generate our own meanings for otherwise objective stimuli by creating unique associations between content and memory (Wittrock, 1974). This process inevitably leads to the application of schemas that are tailored by individuals according to their specific backgrounds, experiences, and beliefs, producing large quantities of variability in learned behaviors and knowledge in the face of "equal" input. Because schemas differ from individual to individual, the mental models generated in response to the same task will infer different outcomes. In the next section, I provide an overview of the research that has aimed to predict programming aptitude and capture neural correlates of code comprehension.

1.2. The Role of Natural Language in Programming Research

In this dissertation, I adopt the view that computer programming, like natural language, relies on complex symbolic representations of goals or ideas, with the primary difference being that it was developed to communicate with machines rather than with other humans. Below, I

provide an overview of how the perceived parallels between programming languages and natural languages have influenced research on programming aptitude regarding its predictors as well as its neural correlates.

1.2.1. Predicting programming aptitude. To date, the predominant way of addressing shortcomings in programming education is to investigate the predictors of successful learning. The reasoning behind this line of research is that understanding the individual characteristics that predict success or failure in the current learning environment is a precursor for understanding why, and for whom, learning to code is challenging. Ultimately, this information might be used to develop new learning environments that promote success in a broader group of students.

Early studies on the predictors of programming, which operated under the assumption that mathematics skills lied at the heart of programming ability, reported positive relationships between mathematics performance in school and performance in a programming course (Alspaugh, 1972; Butcher & Muth, 1985; Konvalina et al., 1983; Petersen & Howe, 1979). Similar inferences have been made regarding the relationship between various cognitive skills and programming ability, including working memory capacity and reasoning skills, which have influenced the content of early programming aptitude tests (see Pea & Kurland, 1983 for a review). However, these proposed predictors also correlated highly with general intelligence, bringing to question whether they were unique predictors of programming ability or byproducts of an individual's overall ability to learn (Pea & Kurland, 1983).

Rather than selecting predictors based on preconceived ideas of what programming success "should" entail, Pea and Kurland (1983) were one of the first researchers to propose that we shift our focus to the *goals* of programming and the cognitive processes that would support them. The goal of programming parallels the goal of natural language comprehension—to extract

the purpose of a larger chunk of code or text by integrating all of the available information at one's disposal (e.g., Fedorenko et al., 2019; Pea & Kurland, 1983). As such, Pea and Kurland (1983) argued that "skilled computer programming is similarly complex and context-dependent" as reading comprehension, "which entails an elaborate body of world knowledge, comprehension, monitoring, inferencing, hypothesis generation, and other cognitive and metacognitive strategies." Others voiced a similar claim by suggesting that natural language ability and programming ability may be related (Kurtz, 1980; Wills, 1982). This line of thinking led researchers to begin considering the shared predictors of natural language learning and programming acquisition, beyond traditional measures related to mathematics skills.

Since the 1980s, more studies have included natural language-related abilities as predictors of programming aptitude to investigate its joint predictive utility with other cognitive abilities, although results have been mixed. For instance, Sauter (1986) found that natural language aptitude and mathematical aptitude predicted different aspects of COBOL knowledge acquisition. Specifically, a student's college verbal entrance exam score was the only variable that contributed significantly to performance on a COBOL syntax test, while a student's college math entrance exam score was the only variable that contributed significantly to performance on logic rules in COBOL. This dichotomy was not present in earlier studies on students learning FORTRAN, which Sauter suggested was the result of the inherent differences between COBOL and FORTRAN; while FORTRAN is mathematically based, COBOL is language-based, hence the reliance on the mechanisms behind natural language processing to learn the syntax.

Shute (1991) extended this body of work by training programming-naive college students on Pascal via an intelligent tutoring system. Following the training phase, participants completed an exam that asked them to identify errors in Pascal programs, order Pascal commands into a

cohesive solution, and generate Pascal code from scratch. 68% of the variance in Pascal learning outcomes was explained by three categories of variables: general knowledge and skills, problem-solving abilities, and learning behaviors. General knowledge and skills consisted of a battery of tests grouped into working memory, general knowledge, information processing speed, technical knowledge, and perceptual speed, but only working memory remained significant following hierarchical regression. This is surprising given that the general knowledge factor primarily consisted of language-related abilities, such as word knowledge, paragraph comprehension, and semantic relations, contradicting prior work (e.g., Sauter, 1986) that implicated the importance of verbal ability in acquiring programming skill.

As research in the programming aptitude field slowly progressed over the past forty years, it became apparent that the variability in results may be due to different predictors that were included across studies, as well as the way programming ability was operationalized. Often, the perspective with which a study approached programming aptitude and acquisition determined the direction of the design, and subsequently, the results. For example, studies that approached programming acquisition from a traditional aptitude perspective have found that academic performance (e.g., course grades or GPA) and standardized measures (e.g., SAT verbal and mathematics tests, or composite scores representing related abilities) contributed to programming course grades (Austin, 1987; Erdogan et al., 2008; Whipkey, 1984). Studies that focused on the characteristics of the *learner* and the *learning process* have linked programming skill to cognitive styles, learning styles, expectations and attitudes towards programming, metacognitive processes (e.g., self-assessments of programming knowledge), and even personality traits (e.g., introversion/extroversion) (Bishop-Clark, 1995; Kagan & Douthat, 1985; Pocius, 1991; Simon et al., 2006; Vu et al., 2000). Quille & Bergin (2018) attempted to reconcile these varied findings

by broadening the scope to predicting the programming success of 692 students from 11 institutions studying different programming languages, such as Java, C#, Python, Processing, Visual Basic, and C++. Their model, which included the three factors of programming self-efficacy, mathematical ability, and number of hours per week spent playing computer games, predicted programming ability at a 67% accuracy, regardless of the demographic and language learned. However, these three predictors were the *only* predictors that they considered, which still introduced biases regarding what programming "should" look like into their measures.

To address this issue, Prat, Madhyastha, Mottarella, and Kuo (2020) launched a study that aimed to predict the learning outcomes of first-time Python learners using resting-state encephalography (rsEEG), in addition to the previously considered predictors of general cognitive abilities (e.g., fluid reasoning, working memory, and inhibitory control), natural language aptitude, and numeracy. By including a task-free index of intrinsic brain functioning, Prat, Madhyastha, Mottarella, and Kuo (2020) adopted a data-driven approach to understanding the cognitive bases of programming acquisition. Results indicated that the various predictors explained 50-72% of variance in learners' three Python learning outcomes: learning rate (speed of progression through Python lessons), programming accuracy on a coding task, and declarative knowledge of Python semantics and syntax. When averaged across the three learning outcomes, general cognitive abilities accounted for the largest amount of unique predictive ability at 34%, which is consistent with prior work demonstrating the importance of holding chunks of code in working memory to guide comprehension and production (Fedorenko et al., 2019; Sauter, 1986). However, other predictors also accounted for unique variance *on top of* general cognitive abilities, including natural language aptitude at 17%, rsEEG at 10%, and numeracy at 2%. This provided evidence that programming aptitude is not solely related to an individual's general

ability to acquire a complex skill, but that it drew from a variety of pre-existing factors that may aid *programming* acquisition more specifically. Crucially, this study supported the notion that mathematics skills are overemphasized during programming acquisition, and that natural language aptitude and neuropsychometric measures may be better indicators of programming outcomes.

Interestingly, the average predictive utility of the reported predictors were not reflective of the pattern in any individual model. For example, the overall unique predictive utility of language aptitude was averaged across 43.1% in learning rate, 8.7% in programming accuracy, and 0% in declarative knowledge. According to these results, natural language aptitude helps an individual progress through the acquisition phase rapidly and is mildly predictive of their ability to *generate* code, but it is not at all predictive of the quality of their declarative programming *knowledge*. By comparison, power in beta and low-gamma frequency bands recorded over right fronto-temporal networks at rest predicted unique variance in learning rate (10%) and declarative knowledge (20.3%), respectively, but did not predict unique variance in programming accuracy. Previous studies have found positive correlations between frontal beta power at rest and learning rate of natural language learning in adulthood (Prat et al., 2016; Prat, Yamasaki, & Peterson, 2019). The Predictive Coding Hypothesis argues that beta oscillations maintain dynamic representations of meaning during sentence comprehension, and that they deploy top-down mechanisms to facilitate comprehension of predicted completions (Lewis et al., 2016). Taken together, these results suggest that programming may draw upon similar top-down mechanisms to *predict* outcomes during programming acquisition as well as code comprehension. However, the nature of these predictions and what types of information an individual may be using to facilitate them has yet to be addressed.

Altogether, Prat, Madhyastha, Mottarella, and Kuo (2020) showed that natural language aptitude and neuropsychological metrics can be jointly leveraged to understand the cognitive processes involved in programming acquisition on a deeper level. If the same rsEEG indicators have been shown to predict learning outcomes in both natural languages and programming languages, this raises the more specific question of whether code comprehension also shares neural substrates with language comprehension.

1.2.2. Shared neural correlates between code comprehension and natural language comprehension. Because programming draws on a wide array of cognitive skills, attempts to identify the neural correlates of programming ability, especially code comprehension, have been similarly mixed. Siegmund et al. (2014) were the earliest researchers to explore the neural correlates of code comprehension using functional magnetic resonance imaging (fMRI). In their experiment, they contrasted comprehension tasks with syntax tasks to determine the unique neural correlates related to understanding code beyond surface-level reading. During the comprehension tasks, participants determined the output to short Java algorithms. During the syntax tasks, participants were asked to locate (but not fix) three syntax errors related to punctuation injected into the same Java algorithms. Every participant completed 12 trials in an fMRI scanner consisting of a comprehension task (60 seconds), followed by a rest period (30 seconds), the syntax task (30 seconds), and a second rest period (30 seconds).

A network of regions were implicated when contrasting the comprehension tasks with the syntax tasks, including (but not limited to) those associated with natural language processing. Specifically, during the comprehension tasks, there was unique activation in the posterior middle temporal gyrus, which is associated with semantic processing at the word level, and the inferior frontal gyrus, which is related to combinatorial aspects in language processing. Siegmund et al.

(2014) argued that the participants needed to process the *meaning* of single words and symbols, and then integrate them into statements to form an overall semantic meaning during the comprehension tasks. Otherwise, participants would not have been able to produce the correct output, which the researchers verified by observing their answers and asking them to describe what each algorithm was doing after the scan. On the contrary, participants could engage in a surface-level search of simple punctuation errors during the syntax tasks, such as mismatched parentheses or missing semicolons, which the researchers argued did not require a full understanding of the code to accomplish. As such, effortful processing related to language comprehension was not necessary. However, one potential confound is that engaging in the comprehension task prior to the syntax task allowed participants to become familiar with each algorithm. Even though Siegmund et al. (2014) argued that the rest periods were sufficient to minimize learning effects, it is still possible that participants did not need to "reread" the algorithms with the same detail that they might have needed to complete the syntax tasks. Altogether, this study provided tentative support for the parallels between natural language and code comprehension that were proposed by Fedorenko et al. (2019), but more work would have to be done to understand what makes code *comprehension* unique from the proofreading of code.

Despite their shortcomings, Siegmund et al.'s (2014) efforts pioneered additional research on the neural representations that support code comprehension and natural language processing. Ivanova et al. (2020) investigated the extent to which code comprehension primarily relies on domain-general executive processes, which are carried out by the multiple demand (MD) system, versus language-specific comprehension processes, which have been linked to a set of left frontal and temporal brain regions collectively called the language system. In the first experiment, they had participants complete a code comprehension task in Python consisting of

three conditions: Python code with English identifiers, Python code with Japanese identifiers, and English sentence versions of those Python problems. In each task, participants were instructed to select the correct output from four options by pressing a button. In the second experiment, the researchers had participants complete a similar code comprehension task in ScratchJr, a visually-based coding language created to teach children programming concepts. Due to ScratchJr being visually-based, there were only two conditions in this experiment: the code condition, and a condition with English sentence versions of those code problems. Prior to the task in both experiments, participants completed two localizer tasks that would identify their unique MD system and language system.

According to participants' fMRI activation patterns, the MD system exhibited strong bilateral responses to code written in Python and ScratchJr, whereas the language system only responded to English sentence problems and weakly to code problems written in Python. As such, Ivanova et al. (2020) argued that, overall, domain-general processes contribute more to code comprehension than language-specific processes. They noted that there was no difference in the activation patterns between Python code with English identifiers and Python code with Japanese identifiers, regardless of individuals' familiarity with Japanese. The researchers speculated that the code problems were simple enough such that an understanding of the identifiers was not crucial to determining the correct output, but they had limited sample sizes as well as vaguely defined groups (no reported Japanese knowledge: N = 16; some familiarity with Japanese: N = 8). Furthermore, if the code problems were simple enough to not require an understanding of the meaning of identifiers, then this brings up the question of whether a deep parsing was necessary in any of the code problems, including the Python problems with English identifiers. In other words, there may not have been a need to integrate semantic meaning with

code syntax to reach the correct output due to the simplicity of the problems. Although the MD system is primarily associated with domain-general processes, it has also been found to be recruited during most tasks that require our attention, including language processing on a more limited scope (e.g., Diachek et al., 2020). It is possible that the participants treated the code problems as simple arithmetic problems or puzzles, which would recruit the MD system, or that the language system was involved in code comprehension (as in the case of Python code), but not as much as when reading natural language text.

In a different study, Liu et al. (2020) attempted to address the same question of whether Python code comprehension elicited consistent neural signatures across individuals that were more similar to symbolic systems like logic and mathematics, which they argued were taught later in life, or a system like natural language that is often obtained from birth. Participants were exposed to two code conditions, and each trial consisted of three phases. In the real code condition, participants first saw a fully defined function that spanned five lines. After a fixation period, the function reappeared, along with an input presented at the bottom that would reference the function. The participants were instructed to mentally derive the output given the function and the input during this phase. After another fixation period, a suggested output would appear, and participants had to indicate whether it was correct by pressing a button. In the fake code condition, participants were first asked to memorize a fake version of a function that was created by scrambling the line order, as well as the content within each line. These fake code conditions were derived from the real functions, but participants did not see real code and fake code that were generated from the same base code. In the second phase, the fake code would reappear, along with a fake scrambled input at the bottom that could not be integrated with the fake function. The participants were instructed to memorize this new fake input. In the third response

phase, a line of "code" would appear, and participants had to indicate whether this line was present in the previous fake function and input screen by pressing a button. In addition to these code tasks, participants also completed localizer tasks to identify the neural substrates that supported their formal logic, symbolic math, and natural language processing.

According to Liu et al.'s (2020) results, a network of left-lateralized regions was activated across individuals during Python code comprehension. This network, which they called the "code comprehension network," included the intraparietal sulcus, select regions in the lateral prefrontal cortex, and the posterior-inferior middle temporal gyrus. Contrasts revealed that this network was more responsive to real code than fake code. When they compared this code comprehension network to the networks implicated in the localizer tasks, the researchers found that it was the most similar to the fronto-parietal system that supported formal logical reasoning (and mathematics to a lesser degree). By comparison, the code comprehension network had less overlap with the fronto-temporal network, which has been traditionally deemed the language network. As such, Liu et al. (2020) suggested that their findings were complimentary to those reported by Ivanova et al. (2020), concluding that code comprehension drew more resources from systems that were not specific to natural language processing. However, they also acknowledged that the regions that *were* shared between code comprehension and natural language—the left inferior frontal and middle temporal gyri, which support high-level linguistic processing such as sentence-level syntax—indicate that the two systems are not completely orthogonal (e.g., Bornkessel-Schlesewsky & Schlesewsky, 2013; Fedorenko & Thompson-Schill, 2014; Hagoort, 2005; Pallier et al., 2011).

Overall, however, these results are contrary to a previous study by Floyd et al. (2017), who were inspired by research suggesting that readability is the most important trait of any

software (Buse & Zimmermann, 2012; Elshoff & Marcotty, 1982; Haneef, 1998; Knight & Myers, 1993; Knuth, 1984; Shull, Rus, & Basili, 2001). Not only did their model predict whether a participant was reading C code or English prose with 79% accuracy solely based on fMRI activity, they found that programming expertise could attenuate the degree to which the neural representations for the two tasks were differentiated. The more proficient an individual was in C, the more overlap there was in their neural representations of C code and English, which conflicts with the findings reported by Ivanova et al. (2020) and Liu et al. (2020). It is possible that C, used by Floyd et al. (2017), and Python, used by Ivanova et al. (2020) and Liu et al. (2020), are unique enough in their syntax constructions such that natural language plays different in their comprehension. However, this would be surprising given that Python was originally designed to be more readable (Scarlett, 2023); as such, one would expect studies on Python to reveal more similar neural representations between code comprehension and natural language compared to other programming languages.

Another possible explanation for this discrepancy is that, similar to predictors of programming aptitude, the neural representations for code comprehension depend on the task demands. For instance, Castelhano et al. (2021) proposed that the reading and mathematics systems have weighted contributions to programming based on the depth of knowledge required to solve the problem at hand. This could be based on the stage of coding itself (e.g., reading, writing, or debugging), as well as the amount of experience that a programmer has with the programming language. If so, it is possible that experts develop fine-tuned cortical representations that are specialized for programming as a result of their knowledge and experience. This would not be a surprising conclusion if we remember that an individual constructs mental models of programming by referencing their past knowledge and existing

skills, which are likely to differ from their peers'. If programmers vary in their general cognitive and natural language skills, then some of those skills will carry more weight in supporting the programming learning of a few individuals more than others, leading to unique learning trajectories, neural representations, and the application of schemas for every person. This could explain (or be a byproduct of) differences in the organization of mental models between experts and novices, which would dictate their approach to programming-related activities.

To fully uncover the "difficulties and complexities faced by students" during programming acquisition (Jenkins, 2002), we must investigate the extent to which their constructed mental models aid or interfere with their ability to complete programming tasks accurately and efficiently. In the next section, I will provide an overview of research on mental models of programming; I argue that knowing what information to draw upon, as well as how to organize the obtained information into a coherent mental model, is part of what separates programming experts from novices—relating new content to past experience does not always lead to efficient learning.

1.3. The Role of Mental Models in Programming Skill Acquisition

1.3.1. Accurate schemas support top-down mental model creation in programming.

Expertise in programming comes from the ability to have "efficiently organized and specialized knowledge schemas" that go beyond a surface-level understanding of language semantics and syntax (von Mayrhauser & Vans, 1994). Such schemas enable a "top-down, breadth-first approach" to all programming tasks and are flexible in their application to allow for varying problem types, contributing to the generation of effective mental models. In particular, expert programmers succeed because they can "reduce variability" of programming tasks "by defining the best way to approach the design task, by supplying a standard of schemas to answer a

question, and by constraining the choices about execution structure to the 'best' solutions" (Rist, 1995). For example, an expert programmer who has encountered a for loop many times will have a template for its execution in which all of the individual pieces are encoded together into one chunk, or schema. By simplifying the task demands into components that they are familiar with, experts rarely need to engage in effortful computation or study code in a bottom-up, unit by unit fashion. Instead, they rely on their schemas to form top-down hypotheses, or *predict* the goal, output, or flow of a program before they have reached the end (Brooks, 1977, 1983; Robins et al., 2003). Such predictions are accurate more often than not, which speeds up processing. Therefore, experts' mental models reflect their ability to combine accurate schemas with effective problem-solving strategies to complete a programming task, rather than the mere parroting of facts. The more experience a programmer has with different code structures, the more examples they have to draw from while solving a task, which expands the depth of their mental models.

1.3.2. Challenges to developing effective mental models of programming.

Programming is a difficult skill to acquire due to the task complexity. Even if novices perform well on exams that test content knowledge, their explanations of *why* an answer is correct would often reveal deficiencies in their understanding of how a programming language is constructed and used (Bayman & Mayer, 1983). This is because novices tend to form misconceptions about foundational programming knowledge; as a result, they retrieve schemas that are inherently inaccurate (Anderson & Jeffries, 1985; Muller et al, 2007; Qian & Lehman, 2017; Spohrer & Soloway, 1986). Due to the constant need to address errors in their comprehension or generated code, novices rely more frequently on a bottom-up approach to programming tasks, which

promotes ineffective mental models related to code execution (Curtis, 1984; Qian & Lehman, 2017; Robins et al., 2013).

Part of novices' misconceptions are tied to the schemas that they have developed based on their preconceived notions of programming, as well as pre-existing knowledge that hinder their learning (Du Boulay, 1986; Pea, 1986; Qian & Lehman, 2017; Simon, 2011). For instance, beginning programmers assume that the computer will know more than what the user tells it to do (Pea, 1986). As a result, novices skip important steps in their code or fail to provide all of the necessary details for the computer to carry out their vision, leading to bugs that disrupt the general flow of program execution (i.e., the order with which statements are evaluated by the computer) (Du Boulay, 1986; Simon, 2011). This misrepresentation of the computer's role in programming is an example of a flawed schema of the relationship between user and computer.

Due to their misunderstanding of a computer's autonomy, novices may form flawed schemas about other programming concepts that relate to program flow. For example, because novices do not always understand the order with which computers evaluate statements, they may scramble the statements needed to assign values to variables (Du Boulay, 1986; Ma, 2007; Sirkiä & Sorva, 2012), attempt to store more than one value in a single variable (Doukakis et al., 2007; Sleeman et al., 1986), or fail to account for the local or global scope of the variable during execution (Fleury, 1991). This can impact the outcome of larger code structures that reference a given variable, such as conditional statements (e.g., if/else) (Green, 1977; Sirkiä, 2012; Sleeman et al., 1986), for loops (Sleeman et al., 1986), and recursion (Kahney, 1983). If a novice relies on flawed schemas during programming comprehension, which involves the parsing of hierarchical structures, they cannot generate mental models that accurately reflect the goal of a task.

Aside from preconceived notions about the computer interface, novices form flawed schemas when they attempt to bridge gaps in their programming knowledge by drawing on their previous experiences with natural language and mathematics (Bonar & Soloway, 1985; Bruckman & Edwards, 1999; Du Boulay, 1986; Miller, 2014). The logical assumption is that basic English vocabulary or mathematics facts will function as schemas by helping a beginner learn similar functions, keywords, or operators in a programming language. However, depending on the programming language, these pre-existing schemas can lead beginners to misrepresent their programming alternatives, resulting in ineffective programming schemas. Some of the most common natural language related errors are related to novices' use of boolean statements and operators; for example, "and" is a boolean operator in many programming languages whose purpose is to evaluate the True/False nature of two statements, which is different from its role as a conjunction in English (Qian & Lehman, 2017). Similarly, introductory programming courses teach beginners to use meaningful variable names to make their code easy to read (e.g., store a list of GPAs in a variable named "GPA"). However, novices may mistake the English meaning of a variable's name with its value, such as believing a variable defined as "GPA" always contains information about grade point averages (Kaczmarczyk et al., 2010; Sleeman et al., 1986). Additionally, novices may mistakenly believe that two elements that contain similar words refer to the same content (e.g., the variables "GPA" and "GPA_college" contain the same information). This is called a reference-point error (Miller, 2014). Mathematics facts can also interfere with a novice's ability to learn programming syntax, with the most common example being the tendency to confuse variable assignment with a computational algebraic expression (e.g., the role of the equal sign "="), or the behavior of integer division that involves a remainder (Clancy, 2004; Doukakis et al., 2007; Qian & Lehman, 2017).

Although many individuals do not formally acquire programming skills until secondary education, novices' programming misconceptions can begin as early as kindergarten (Mertala, 2019; Mladenović et al., 2018; Pea & Kurland, 1983). As such, flawed programming schemas can become as difficult to unlearn as bad habits; it takes many repetitions of the correct way to do something before the old way is overwritten (Pea & Kurland, 1983). However, if a novice's learning experience is inherently filled with errors, they may not know how to course correct without proper guidance, even with repeated practice. Therefore, if we wish to improve programming education, interventions should be put in place that focus on giving learners the building blocks to support effective mental model construction.

1.3.3. Evidence for accurate schemas supporting mental model construction in novices. As discussed in the previous section, computer programming is a symbolic system that benefits from well-developed schemas. As such, providing novices with techniques to create accurate schemas of their own should promote programming skill acquisition. There was already evidence that giving learners appropriate "advance organizers" led to better comprehension of complex technical information in history, mathematics, and science (Ausubel, 1960; Lesh, 1976; West & Fensham, 1976). "Advance organizers" include general concepts and ideas that can be used to glean an understanding of new material (Ausubel, 1960). They can be thought of as a schema that encourages learners to connect information with existing knowledge that they already have, in ways that actually support learning. Crucially, such schemas are most useful for individuals who completely lack experience and need to use the information in novel problem-solving contexts. Inspired by these results, Mayer (1975) gave some of his subjects concrete models that broke down basic computer programming functions prior to having them read manuals on a BASIC-like language. The models included analogies that explained the role of

input, output, memory, and executive control via a series of figures; for example, executive control was represented using a recipe that included an arrow to indicate the line that was currently being followed, or executed. Subjects who were given the models performed better on a transfer test of the learned material than subjects who were not given the same models, but only on problems that required generating answers for novel scenarios. Both groups performed similarly on questions that were similar to the material directly presented in the manual as well as extremely complex looping problems, showing that the benefits of the concrete model depended on the task complexity. In a follow-up study, Mayer (1976) showed that individuals who were given the model before reading the manual performed better than individuals who were given the model after reading the manual. Subjects were able to directly utilize the organizing information to support their learning, similar to the readers in Bransford and Johnson's (1972) study whose recall was facilitated by being provided with context for a vague paragraph. These results have been replicated in studies on other programming languages, such as SEQUEL (Mayer, 1980) and BASIC (Bayman & Mayer, 1988), in which providing students with information that emphasized the conceptual reasoning behind the language's syntax reduced the number of misconceptions they would develop.

Evidently, if students are guided correctly, they can acquire programming skills that allow them to solve novel problems. Because novices' struggles are directly tied to the information they have available during learning, there is a lot of variability in what they can do wrong. Conversely, this means that people can rely on different skill sets on their paths to reach the same destination. Therefore, the key to understanding the difficulties in programming acquisition lies in what happens during *real-time* construction of a mental model. To directly address how programmers of varying proficiencies rely on their schemas during programming,

we need a tool that can index code comprehension while it unfolds. In the next section, I argue that this can be achieved using the event-related potential (ERP), and I provide an overview of its role in uncovering mental models in other domains before proposing how this method can be applied to studying real-time programming.

1.4. The Event-Related Potential as a Neural Index of Mental Model Usage

Computer programming involves real-time processing and manipulation of incoming information, with a programmers' efficiency of doing so being tied to their expertise in the area. One of the most common ways to study psychophysiological correlates of cognitive activity is the event-related potential (ERP). By placing electrodes on the scalp, we can measure electroencephalograms (EEG), which is amplified electrical activity generated by neurons in the brain. EEGs that are time-locked to specific events of interest become ERPs, or evoked potentials to stimuli (Luck, 2005). Walter and colleagues discovered the first cognitive ERP component in 1964, the contingent negative variation, which showed that subjects would exhibit a large negative voltage while preparing to respond to an upcoming target *before* it was shown to them. This revolutionary finding indicated that ERPs could be used to index individuals' *prediction* of future stimuli based on their expectations, not simply their reactionary responses to sensory input.

ERPs capture changes in electrical activity on a millisecond-by-millisecond basis, which make them useful for studying the mental models associated with real-time processing of information. One of the fields that has benefited the most from the ERP approach is the study of natural language processing. Although behavioral measures such as vocabulary and grammar tests can capture proficiency on-paper, the ERP method allows researchers to investigate how language users integrate input, resolve ambiguities, and predict with precision in real-time.

1.4.1. The N400 component in natural language. Native speakers of natural languages exhibit distinct brain responses to violations of semantics (N400 component) and syntax (P600 component) (Osterhout & Nicol, 1999; Swaab et al., 2012). The N400 component is a negative-going wave that peaks in amplitude around 400 ms after the presentation of a stimulus (Swaab et al., 2012). It can be observed in response to words, sentences, or large chunks of discourse presented in various modalities, including written text, speech, and sign language, in which there is an unexpected stimulus given the preceding semantic context (Kutas et al., 2006). The earliest documented example comes from Kutas and Hillyard's 1980 study that asked native English speakers to read sentences that ended with a congruent ending (e.g., *I shaved off my mustache and beard.*), a semantically improbable word (e.g., *He planted string beans in his car.*), or a semantically anomalous word (e.g., *I take my coffee with cream and dog.*). The semantically anomalous endings elicited the most prominent N400 effects (largest change in amplitude), with semantically improbable endings eliciting weaker, but visibly different N400 effects relative to congruent endings. Despite initially being thought of as a marker for linguistically-motivated semantic incongruity, the N400 component is now believed to reflect meaning processing more broadly (Kutas & Federmeier, 2011; 2014). In particular, the N400 component is thought to be an index of the ease with which we *retrieve* stored conceptual knowledge from semantic memory; the more the context aids in retrieval, the smaller the N400 amplitude (Kutas & Federmeier, 2014). Others have proposed that the N400 component solely represents the ability to integrate semantic information into a unified interpretation (Hagoort, 2005), but this view has largely been debunked (Aurnhammer et al., 2021; Delogu et al., 2019; Lau et al., 2008).

1.4.2. The P600 component in natural language. The N400 component is distinguishable from the P600 component, which is more varied in its function as well as its

interpretation. The P600 component is a positive-going wave that peaks in amplitude around 600 ms after the presentation of a stimulus (Swaab et al., 2012). It was first discovered in response to words that were incongruous with the expected syntax structure in garden path sentences, which are ambiguous in their resolution and require more effort to understand (e.g., *The broker persuaded to sell the stock was sent to jail.*) (Osterhout & Holcomb, 1992). Additional studies showed that the P600 component is also sensitive to outright structural violations, such as phrase-structure violations, gender and case marking, and verb tense violations (Friederici et al., 1993, 1996; Hagoort et al., 1993; Neville et al., 1991). Osterhout & Nicol (1999) demonstrated that the P600 component elicited by syntax violations (e.g., *The cats won't eating the food.*) is distinct from the N400 component elicited by semantic violations (e.g., *The cats won't bake the food.*); words that were incongruous in both semantics and syntax would elicit both components (e.g., *The cats won't baking the food.*). However, under certain circumstances, the P600 component can appear in response to semantic violations between verbs and their arguments (Kuperberg, 2007). This typically happens when the thematic role of a verb as implied by the provided context is inconsistent with the role that it usually inhabits. For example, the word "devouring" in the sentence "*The meal was devouring…*" elicited a P600 effect but not an N400 effect, presumably because meals are usually the object being devoured instead of the agent doing the devouring (Kim & Osterhout, 2005). This "semantic attraction" effect has been replicated in studies in which the semantic association between the verb and its argument is not as strong (e.g., *Every morning at breakfast the eggs would plant…*) (Kuperberg, 2007). Therefore, the P600 component is reflective of predictive structure building, with the amplitude being indicative of integration effort as a result of resolving structure violations or unexpected structural continuations (Aurnhammer et al., 2021, 2023; Swaab et al., 2012).

1.4.3. The N400 and P600 components as indices of natural language proficiency.

The N400 and P600 components do not present the same across individuals (Swaab et al., 2012; Tanner et al., 2014); as a result, variability in these components have been used to study individual differences in cognitive processing. In the realm of natural language processing, the development of the N400 and P600 components have become reliable indicators of second language acquisition. In a study that asked subjects to read German sentences containing either well-formed verbs or incorrectly inflected verbs, third year English-speaking learners of German who were more advanced in their study exhibited a P600 effect similar in amplitude to native German speakers (McLaughlin et al., 2010). On the contrary, first year English-speaking learners of German exhibited weak N400 and P600 effects to the incorrectly inflected verbs. Additional analyses revealed that the grand average waveform for first year learners did not represent most individuals who made up that group; instead, first year learners could be separated into individuals who predominantly exhibited an N400 effect and individuals who predominantly exhibited a P600 effect. These results suggested three key findings. First, learners of a second language initially exhibit an N400 effect to structural violations; second, learners can transition to displaying native-like brain responses with proficiency; and third, learners achieve native-like brain responses at varying rates, perhaps reflective of the time it takes to internalize syntax rules. These findings were confirmed in a longitudinal experiment that followed a group of English-speaking French learners in their first year of study (McLaughlin et al., 2010). To French sentences containing structural violations, the learners primarily exhibited an N400 effect, then no clear effects, then a P600 effect after 4 weeks, 16 weeks, and 26 weeks of French instruction, respectively. However, there were also individual differences in the rate at which learners progressed from primarily displaying N400 effects to displaying P600 effects, with some

individuals exhibiting P600 effects from the start, and others still exhibiting N400 effects by the end of instruction. Altogether, these results demonstrate that language proficiency leads to different brain responses to structural anomalies, and learners follow their own time trajectories in achieving native-like responses.

1.4.4. Using the N400 and P600 components to investigate comprehension and expertise in non-linguistic domains. Even though the N400 and P600 components were discovered in response to semantic and syntactic anomalies in natural language, they have been documented in other domains involving the manipulation of meaning and form. Relative to correct solutions, incorrect solutions to mathematics problems elicit N400 effects that are indistinguishable from the N400 component elicited by semantically incongruous words in written sentences (Galfano et al., 2004; Niedeggen et al., 1999), while deliberate violations of mathematics rules elicit P600 effects (Núñez-Peña & Honrubia-Serrano, 2004). ERPs have also demonstrated individual differences in perceptions of semantic misalignment during mathematical modeling, or the process by which people reason using mathematical representations of real-world situations (Guthormsen et al., 2015). Individuals who undergo mathematical modeling rely on schemas that dictate the relationship between operators and objects; for example, categorically related objects can be added together (e.g., *roses* + *tulips*), and functionally related objects can be divided into each other (e.g., *tulips / vases*) (Bassok et al., 1998). Non-modelers do not engage in this type of semantic alignment during mathematical processing and instead judge operations solely based on the numerical outcome. After using responses to a mathematics acceptability judgment task in which some statements were semantically aligned (e.g., *Twelve forks plus three spoons equals fifteen*) and some were misaligned (e.g., *Twelve forks divided by three spoons equals four*), Guthormsen et al. (2015)

found that modelers and non-modelers displayed different brain responses to varying parts of the statements. In non-modelers, there were no differing brain responses to the second object word (*spoons*) between the aligned and misaligned conditions, whereas modelers displayed a P600 effect to the misaligned condition relative to the aligned condition. Non-modelers also did not differ in their brain responses to the number at the end of mathematically correct statements, whereas modelers displayed an N400 effect to the misaligned condition (*four*) relative to the aligned condition (*fifteen*). Therefore, even though both statements were correct in terms of their mathematical answers, modelers treated the misaligned statements as violations in meaning and form. These findings show that for some individuals, mathematical thinking is not independent of semantic knowledge. Similar effects of meaning and form have been found when comparing expert musicians' and musically-naive participants' ERP responses to musical stimuli. For instance, musicians exhibit N400 effects to offkey and unexpected melodies (Calma-Roddin & Drury, 2020), as well as P600 effects to violations of harmony and key progressions (Besson & Macar, 1987; Janata, 1995; Patel et al., 1998).

Altogether, the presence of the N400 and P600 components in response to anomalies of meaning and form in natural language, mathematics, and music is an indication that prediction violations of symbolic systems can be measured. The magnitude of these components indicates an individual's expertise in the domain; the stronger the schema, the more egregious the violation, leading to a larger effect. Therefore, if programmers use schemas to support their mental models during programming tasks, then we can measure their ERPs to well-formed and anomalous code, using the results to understand how meaning and form are processed. Individual differences in ERPs will provide insight into the mental models constructed by programmers of varying skill sets.

1.5. The Current Study

The goal of this dissertation is to investigate mental models of code comprehension by measuring programmers' ERP responses to violations of meaning and form. Python is selected as the programming language of study due to the fact that it was developed to have a "simple syntax, which makes it easily readable and extremely user- and beginner-friendly" (Scarlett, 2023). This feature is believed to drive Python's rapid growth in popularity. As of 2022, Python was the second most-used programming language on GitHub, with its usage growing more than 22% on a yearly basis (Scarlett, 2023). Understanding how programmers process the meaning and form of Python code would provide insight into the comprehension mechanisms used by a large portion of the programming community.

Participants' brain responses are predicted to be similar to brain responses to meaning and form violations in other symbolic systems, such as natural language processing. Furthermore, if there are individual differences in the strength of the schemas used by programmers, they should display different brain responses to the same code anomalies when separated by these differences. Additionally, I propose that programmers' ERP responses to violations of semantics and structure in code will bring us closer to understanding how their mental models for programming languages may be organized.

1.5.1. Predictions.

1.5.1.1. Group level responses to meaning and form code violations. If programming languages are processed similarly to other symbolic systems like natural language (Fedorenko et al., 2019) and mathematics, programmers should show distinct neural components to meaning and form violations in code. Specifically, programers are predicted to exhibit the N400

component to meaning (semantic) violations and the P600 component to form (structure) violations.

1.5.1.2. Individual differences in responses to meaning and form code violations.

According to prior research, expert programmers have stronger mental models that allow them to process and predict the resolution of code chunks before reaching the end of the program (Robins et al., 2013). Therefore, compared to novice programmers, expert programmers will be more sensitive to structural anomalies because they will have more rigid schemas in place as a result of their more developed mental models. In line with this prediction, experts will display more prominent P600 effects to violations related to form.

Natural language has been shown to facilitate the processing of semantic information (Anderson, 1977; Barnitz, 1986; Carrell, 1984). Therefore, English proficiency, measured as English reading comprehension ability, is also predicted to be an indicator of individual differences in brain responses. Specifically, individuals with higher English reading comprehension ability will be more sensitive to semantic anomalies due to their more concrete expectations for semantic consistency, exhibiting more prominent N400 effects to violations related to meaning.

Method

2.1. Participants

Sixty-two right-handed individuals with normal or corrected-to normal vision and no history of significant head injury or epilepsy were recruited for participation in this study. All participants had a minimum of the equivalent of one academic quarter's worth of Python instruction, either through a live course taught by an instructor or self-taught via an online course. One individual did not return for the ERP session and was removed from all analyses. Of

the remaining 61 individuals, 16 exceeded the maximum 25% rejection rate threshold for their averaged ERPs and were removed from all analyses. Detailed rejection criteria are included in Data Analysis (section 2.4). The final sample consisted of 45 English-speaking participants (23 female, 21 male, 1 other, aged 18-33 years) from various natural language backgrounds. One participant wore hearing aids during EEG recording. All experimental procedures were approved by the University of Washington Institutional Review Board. Participants gave informed consent prior to the start of the experiment and were monetarily compensated for their time.

2.2. Materials

2.2.1. Python proficiency measures. Participants completed two measures of Python proficiency, a self-report questionnaire modeled after the Language Experience and Proficiency Questionnaire (LEAP-Q; Marian et al., 2007—see English proficiency section for additional detail) and an objective multiple-choice Python knowledge test. Descriptives as well as additional Python demographic information are presented in the Results.

2.2.1.1. Self-report: Python Experience and Proficiency Questionnaire (PEAP-Q). The PEAP-Q was developed by adapting questions in the LEAP-Q (Marian et al., 2007) to ask about Python experience. In addition to their Python acquisition history, participants self-reported their Python proficiency, daily usage of Python, and history of Python usage in various environments. Overall self-rated Python proficiency was calculated as the average of their reported proficiencies in reading, debugging, and writing Python code on a scale from 0 (no skills) to 10 (expert proficiency). Daily usage of Python was reported as the average of their estimated daily minutes reading, debugging, and writing Python code at the time of the experiment, using the past month as a reference point. History of Python usage was provided as three separate

measures: the number of months using Python in a work environment, school environment, and personal environment (i.e., for personal pleasure).

2.2.1.2. Python knowledge test. Participants completed a 72-item multiple-choice test as a quantitative measurement of their Python proficiency (Prat, Madhyastha, Mottarella, & Kuo, 2020). Half of the questions measured semantic knowledge, such as the purpose of functions and operators (e.g., "What does the print() function do?"). The other half measured knowledge about syntax, or structural rules of Python (e.g., "Why won't the following code compile?").

This measure was developed as part of another study that assessed learners' Python knowledge following weekly lessons in the Python 2 course on Codecademy (Kuo et al., 2022). As such, the questions were created according to the material covered, including, but not limited to strings, conditionals, functions, lists, dictionaries, and loops (Codecademy, n.d.). Python proficiency was quantified as a percentage accuracy score by dividing the total number of items correct by 72 total possible items.

2.2.2. English proficiency measures. Participants completed two measures of English proficiency, a self-report questionnaire and an objective multiple-choice English reading test. Descriptives from these measures are presented in the Results.

2.2.2.1. Self-report: Language Experience and Proficiency Questionnaire (LEAP-Q). The LEAP-Q is a questionnaire designed to index individuals' natural language experiences (Marian et al., 2007). English proficiency was of particular interest in this experiment. In addition to their English acquisition history (e.g., age of acquisition, length of experience), participants self-reported their current proficiency in reading, speaking, and understanding English on a scale from 0 (no proficiency) to 10 (native proficiency). Overall self-rated English proficiency was calculated as the average of their reported proficiencies in reading, speaking,

and understanding English. Participants also provided their daily usage of English, which was reported as three separate measures: the estimated percentages of a typical day spent reading, speaking, and listening to English compared to other languages they know.

2.2.2.2. Nelson-Denny Reading Comprehension. Participants completed the comprehension subtest of the Nelson-Denny Reading Test (NDRT) as a quantitative measurement of their English reading comprehension ability (Brown, 1960). The NDRT is a reading test created for, and normed on, college-aged readers. First, participants read a passage in silence for one minute, and their progress was converted into a reading rate score. Second, participants have 19 minutes to read seven unique English passages and answer multiple-choice reading comprehension questions related to the content. There were 38 questions across the seven passages. Although the NDRT provides tables that convert raw scores to percentiles, these norms were modeled on typical native English speaking adults, split by gender, which may introduce a biased interpretation of an individual's score for this sample. As such, English reading comprehension ability for the present study was quantified as the total number of items correct out of 38, converted into an accuracy score out of 100%.

2.2.3. ERP stimuli. Lines of Python code were created by crossing semantic plausibility and structural validity in a two by two design. A unique global variable preceded every line of code, which provided thematic context for an individual trial. Global variables could be strings, integers, floats, lists, or dictionaries. Each line of code contained a single unit of code that was either semantically plausible or semantically implausible given the global variable, as well as either structurally valid or structurally invalid according to Python syntax rules. This resulted in four different code conditions: 1) well-formed, 2) semantically implausible, 3) structurally

invalid, and 4) doubly anomalous (semantically implausible and structurally invalid). All lines of code spanned five to nine unit lengths with the violation position occurring at two, three, or four.

Each line of code was written according to the syntax rules of Python version 3.0. In order to provide variability in stimuli, two code structures were used in this experiment: for loops and list comprehensions. Violations were created by manipulating one of two code types: variables or keywords. Variables are placeholder words (i.e., iterators) that parse through each item of an iterable, such as a list or a dictionary. Although variables can be named using any combination of letters and numbers, it is common practice to give them a label that is thematically consistent with the object being iterated through. For example, if provided with the list *pets = ["dog", "cat", "hamster"]*, a variable named *animal* rather than *fruit* would be more appropriate for iterating through each list item ("for every *animal* in the list *pets*…" versus "for every *fruit* in the list *pets*…"). Variables cannot be attached to operators or symbols, such as quotations that are used to signify a string. As such, variables can be manipulated to be semantically implausible, structurally invalid, or doubly anomalous (Table 1). On the contrary, keywords are reserved words that have specific roles, and they cannot be used as variable names, function names, or other identifiers. The present study manipulated two keywords, *if* and *in*, by replacing them with other English words that were either approximate synonyms or semantically dissimilar. For example, the keyword *in* could be replaced with *within* (an approximate synonym) or *under* (semantically dissimilar). However, because keywords are built-in, they cannot be manipulated to be semantically implausible while remaining structurally valid within the Python syntax rules. As such, keywords can only be manipulated to be structurally invalid or doubly anomalous (Table 1). In order to accommodate for this imbalance, additional lines of

code that manipulated the variable were written to allow for a balanced, fully crossed two by two design.

For the lines of code in which it was possible, the four versions corresponding to each condition were distributed across four experimental lists, such that each list only had one version of each line of code. Participants saw 160 lines of code, with 40 lines from each condition. In the well-formed, structurally invalid, and doubly anomalous conditions, the 40 lines of code were split between 20 manipulations of variables and 20 manipulations of keywords. In the semantically implausible condition, all 40 lines of code were manipulations of variables. Each list was divided into 4 blocks of 40 sentences each, and each block contained 10 lines of code from each condition. The 20 additional lines of code written to round out the semantically implausible condition were included in each list. Lines of code were pseudo-randomized within each list, and list assignment was pseudo-randomized across participants.

Table 1.
Stimulus examples.

Condition	Variable Example	Keyword Example
Well-Formed	for animal in pets: print(animal)	for tree in forest: print(tree)
Semantically Implausible	for fruit in pets: print(fruit)	N/A
Structurally Invalid	for "animal" in pets: print(animal)	for tree within forest: print(tree)
Doubly Anomalous	for "fruit" in pets: print(fruit)	for tree under forest: print(tree)

Note. The critical piece of code for ERP averaging is underlined. The global variables shown before each trial were pets = ["dog", "cat", "hamster"] (Variable Example) and forest = ["oak", "pine", "maple"] (Keyword Example).

2.3. Procedure

Participants took part in two sessions, each lasting no more than two hours. With the exception of one individual, all participants completed the experiment in the same session order. Session 1 was administered over videoconference, during which participants completed all questionnaires and tasks related to demographics, natural language background, and programming experience. During Session 2 on a separate day, participants judged individual lines of Python code for acceptability while electroencephalogram recordings were obtained. After being seated in a desk chair in front of a CRT monitor, participants were instructed to relax and minimize movements and eye blinks while silently reading the stimuli in their minds. Each trial consisted of the following events: Participants were given 15 seconds to read the global variable to be referenced in the upcoming line of code, or they could proceed earlier by clicking a mouse button. Following a 1000 ms fixation cross and 200 ISI, the line of code appeared in the center of the screen one unit at a time, as defined by blank space, at a presentation rate of 700 ms and 200 ms ISI. Due to the novelty of presenting code in this format, a pilot study was run to determine the presentation rate that would maximize performance on the task and minimize working memory demands. The slower presentation rate chosen is standard for ERP studies of a second language, for both native speakers and learners (Foucart & Frenck-Mestre, 2011, 2012; Tanner et al., 2013, 2014). After the line of code was finished displaying, a “Yes/No” screen followed, during which participants had 30 seconds to give their acceptability judgment by pressing one of two mouse buttons. “Yes” corresponded to lines of code that were acceptable, and “No” corresponded to lines of code that were unacceptable. Participants were asked to use their own criteria for what they considered to be “acceptable” and to keep this criteria consistent throughout the session. The order of the “Yes/No” response buttons (left/right) was pseudo-

randomized across participants. Once a response was given, a "READY?" prompt appeared, and participants would click either mouse button to begin the next trial.

2.4. Data Analysis

2.4.1. Code acceptability judgment task. Behavioral performance on the ERP task was assessed via acceptability rates, which were calculated as the percentage of trials that participants considered to be "acceptable," i.e., responded with "Yes" during the end-of-code judgment task. A two (semantic plausibility) by two (structural validity) repeated-measures ANOVA was used to assess differences in acceptability judgments between code conditions for all participants. The Greenhouse-Geisser correction for inhomogeneity of variance was applied to all repeated measures data. In such cases, the corrected statistics, including the *p*-value, are reported. In instances where follow-up analyses were required, the adjustment method chosen for multiple comparisons is reported in-text, along with corrected statistics. Unless otherwise specified, familywise alpha = .05 for all reported analyses.

2.4.2. ERPs.

2.4.2.1. EEG acquisition. Continuous EEG was recorded from 32 scalp electrodes placed in International 10-20 system locations attached to a Biosemi elastic cap (Jasper, 1958). Eye movements and blinks were monitored by two electrodes placed beneath the left eye and to the right of the right eye. Electrodes were referenced to an electrode placed over the left mastoid during recording, then subsequently re-referenced to the average of two electrodes placed over the left mastoid and right mastoid during pre-processing. EEG signals were amplified with a bandpass filter of 0.01 to 30 Hz by a Biosemi bioamplifier system. Impedances at scalp electrodes were held below 50 Hz. Continuous analog-to-digital conversion of the EEG and stimulus trigger codes was performed at a sampling frequency of 200 Hz.

2.4.2.2. EEG cleaning. For each participant, epochs of EEG signal were segmented around the critical unit of code from -100 ms to 1205 ms. Epochs were removed from ERP averaging if they contained changes within a sliding 200 ms window that were greater than 100 μV in the Cz electrode. Trials characterized by excessive eye movement, muscle artifact, and alpha were further removed prior to averaging; these were epochs that showed voltage steps more extreme than -65 μV or 65 μV within any electrode. Participants who had a >25% artifact rejection rate were removed from analysis altogether, resulting in the final sample of 45 individuals. The average rejection rate for participants who were included in the final analyses was 6.8% (by condition: well-formed: 7.5%, semantically implausible: 6.4%, structurally invalid: 6.9%, doubly anomalous: 6.4%). Due to a malfunction in the recording equipment, two participants in the final analyses had fewer than 160 trials (159 and 158 trials); both individuals averaged rejection rates of 3.8%.

2.4.2.3. EEG analysis. ERPs time-locked to the onset of the critical unit of code in each line of code were averaged offline at each electrode site in each condition. The epochs that went into ERP averaging were base-line corrected from -100 ms to 0 ms. As is standard, the following time windows were chosen for analysis: 300-500 ms (N400) and 500-800 ms (P600), with separate analyses conducted for each window. All trials (both acceptable and not acceptable judgment responses) were included in the final analysis. This decision was made for two reasons: First, previous research has shown that neural sensitivity to anomalies sometimes precedes behavioral sensitivity in L2 learners (e.g., McLaughlin et al., 2004), and second, the definition of "acceptable" was intentionally made ambiguous so that learners could decide how to treat semantically anomalous trials. This made deciding whether trials were correct or incorrect difficult in some conditions. Repeated-measures ANOVAs with two levels of semantic

plausibility (semantically plausible, semantically implausible) and two levels of structural validity (structurally valid, structurally invalid) were used to assess differences in scalp amplitudes between code conditions for all participants. Analyses focused on data collected from midline sites (Fz, Cz, Pz) as we had no hypotheses that were centered on scalp topography. Electrode was included as an additional within-subjects factor (three levels). The Greenhouse-Geisser correction for inhomogeneity of variance was applied to all repeated measures data. In such cases, the corrected statistics, including the *p*-value, are reported. In instances where follow-up analyses were required, the adjustment method chosen for multiple comparisons is reported in-text, along with corrected statistics. Unless otherwise specified, familywise alpha = .05 for all reported analyses.

2.4.3. Between-group analyses: Python expertise and English reading comprehension ability. To investigate the influence of Python expertise and English reading ability on acceptability judgments and neural responses, participants were split into two groups in two manners: once according to Python expertise, and a second time according to English reading comprehension ability.

2.4.3.1. Python expertise group definition. Python knowledge test accuracies were used to assign participants to two groups: Python Experts ($N = 27$) and Python Novices ($N = 18$). Participants who scored 75% or more on the Python knowledge test were considered Experts, whereas participants who scored less than 75% were considered Novices. 75% corresponds to a 2.0 on a standard 4.0 GPA scale, which is the lowest passing grade allowed for some courses with more stringent grading policies at the University of Washington.

2.4.3.2. English reading comprehension ability group definition. Because our participants were primarily college students with high English proficiency, we used NDRT

accuracies to sort participants into two groups via a median split. Four participants who scored equal to the median were removed from the group analyses related to English reading comprehension ability. Participants who scored greater than the median were considered English-High individuals ($N = 20$), whereas participants who scored less than the median were considered English-Moderate individuals ($N = 21$).

2.4.3.3. Between-group data analyses. To follow up the whole-group analyses, two additional, separate sets of ANOVAs were carried out on the acceptability judgments, one with Python expertise entered as a between-subjects factor (experts vs novices, N total = 45), and another with English reading comprehension ability entered as a between-subjects factor (high vs moderate, N total = 41). Similar additional analyses were conducted on the ERPs for the 300-500 ms and 500-800 ms time windows. Main effects did not change as a result of including Python expertise or English reading comprehension ability as a between-subject factor for any of the additional analyses; as such, only effects related to Python expertise and English reading comprehension ability are reported in detail.

2.4.4. Effect of code type analyses. To investigate whether programmers process Python code differently depending on the role of the item being manipulated, the present study also analyzed the effect of errors on two types of code: variables, which are flexible and defined by the programmer, and keywords, which are reserved English words that are inflexible and serve specific roles in the Python programming language. Because semantic plausibility could not be fully crossed with structural validity for keywords, comparisons of effects for variables vs keywords focused on the well-formed condition, the structurally invalid condition, and the doubly anomalous condition for the acceptability judgment task as well as the ERPs. As such, a repeated-measures ANOVA with three levels of condition (well-formed, structurally invalid, and

doubly anomalous) and two levels of code type (variables, keywords) was used to assess differences in acceptability judgments. A repeated-measures ANOVA with three levels of condition (well-formed, structurally invalid, and doubly anomalous) and two levels of code type (variables, keywords) was used to assess differences in ERPs at the Cz electrode site only. For all analyses involving code type, the methods utilized for correcting inhomogeneity of variance as well as adjusting for post-hoc multiple comparisons were the same as in the analyses with all trials combined.

2.4.4.1. Analyses for group differences in the effect of code type. Similar to the analyses with all trial types combined, two additional, separate sets of ANOVA analyses were carried out on the acceptability judgments, one with Python expertise as a between-subjects factor (experts vs novices, *N* total = 45), and another with English reading comprehension ability as a between-subjects factor (high vs moderate, *N* total = 41). These additional analyses were also conducted on the ERPs for the 300-500 ms and 500-800 ms time windows. Main effects did not change as a result of including Python expertise or English reading comprehension ability as a between-subject factor; as such, only effects related to Python expertise and English reading comprehension ability are reported in detail for these analyses.

Results

First, whole-group and group-split descriptives are presented for Python proficiency measures and English proficiency measures. Then, behavioral task performance and ERPs are presented together for the whole group, followed by Python Experts vs Python Novices, then English-High individuals vs English-Moderate individuals. Analyses comparing variable trials to keyword trials are presented at the end.

3.1. Python Proficiency Descriptives

Descriptive statistics for all Python-related experience and proficiency measures can be found in Table 2. Performance on the Python knowledge test yielded 27 Python Experts and 18 Python Novices (Fig. 1). Accuracy on the Python knowledge test correlated with participants' overall self-rated Python proficiency, $r(43) = .69$, $p < .001$ (Fig. 2).

Table 2.
Descriptive statistics for Python-related experience and proficiency measures.

Python Measure	Group	Mean	SD	Min	Max
Number of months since first exposure to Python	Whole Group	29.22	26.85	3.00	120.00
	Experts	39.63	28.31	4.00	120.00
	Novices	13.61	14.48	3.00	60.00
Including Python, number of programming languages with prior experience (1-5)	Whole Group	3.07	1.48	1.00	5.00
	Experts	3.63	1.24	2.00	5.00
	Novices	2.22	1.44	1.00	5.00
Overall self-rated Python proficiency (0-10)	Whole Group	5.29	2.32	1.33	9.33
	Experts	6.35	1.86	2.00	9.33
	Novices	3.70	2.05	1.33	7.67
Reading proficiency	Whole Group	5.64	2.24	2.00	10.00
	Experts	6.59	1.80	2.00	10.00
	Novices	4.22	2.10	2.00	8.00
Debugging proficiency	Whole Group	5.02	2.57	1.00	9.00

	Experts	6.04	2.23	2.00	9.00
	Novices	3.50	2.33	1.00	8.00
Writing proficiency	Whole Group	5.20	2.46	1.00	9.00
	Experts	6.41	1.91	2.00	9.00
	Novices	3.39	2.06	1.00	7.00
Overall daily usage of Python (minutes)	Whole Group	27.22	40.74	0.00	140.00
	Experts	41.05	47.09	0.00	140.00
	Novices	6.48	12.11	0.00	36.67
Daily minutes reading Python	Whole Group	32.96	54.92	0.00	240.00
	Experts	49.19	64.99	0.00	240.00
	Novices	8.61	17.39	0.00	60.00
Daily minutes debugging Python	Whole Group	19.58	29.37	0.00	120.00
	Experts	29.67	34.02	0.00	120.00
	Novices	4.44	7.65	0.00	20.00
Daily minutes writing Python	Whole Group	29.13	49.48	0.00	240.00
	Experts	44.30	58.65	0.00	240.00
	Novices	6.39	12.70	0.00	45.00
Number of months spent using Python in a work environment	Whole Group	8.27	15.03	0.00	60.00
	Experts	12.22	17.58	0.00	60.00

	Novices	2.33	7.08	0.00	30.00
Number of months spent using Python in a school environment	Whole Group	13.18	20.72	0.00	108.00
	Experts	18.00	23.31	0.00	108.00
	Novices	5.94	13.70	0.00	60.00
Number of months spent using Python in a personal environment	Whole Group	9.78	22.37	0.00	120.00
	Experts	14.70	27.93	0.00	120.00
	Novices	2.39	2.48	0.00	10.00
Python knowledge test (0-100%)	Whole Group	73.09	13.91	30.56	94.44
	Experts	81.69	5.17	75.00	94.44
	Novices	60.18	12.90	30.56	73.61

Note. Sample sizes: Whole Group ($N = 45$); Experts ($N = 27$); Novices ($N = 18$)

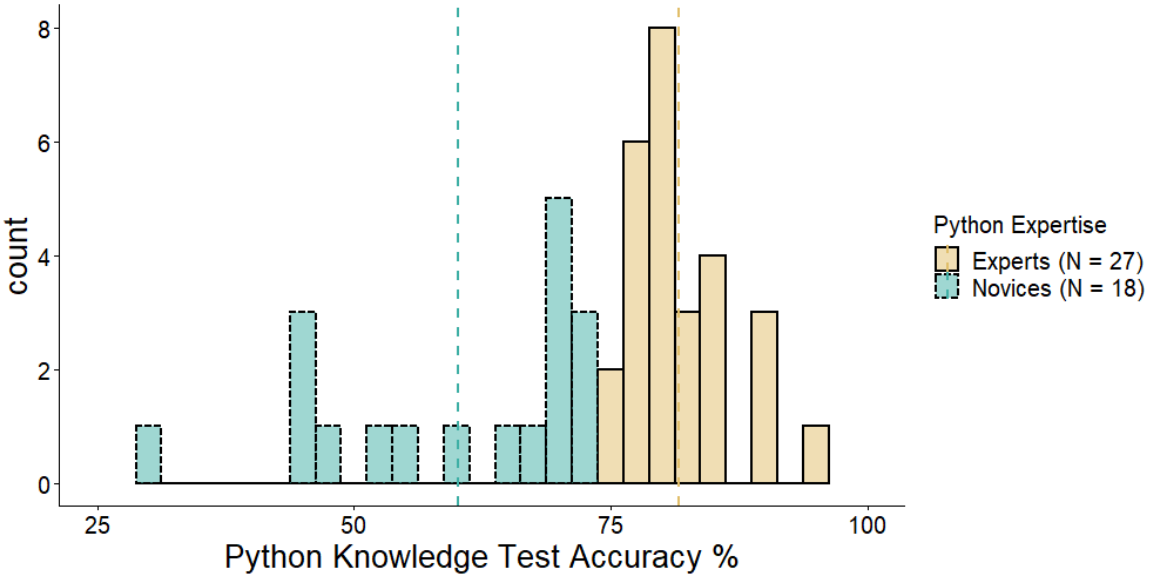

Fig. 1. Distributions of Python proficiency, as measured by accuracy on the Python knowledge test. Dashed vertical lines denote the means per sub-group.

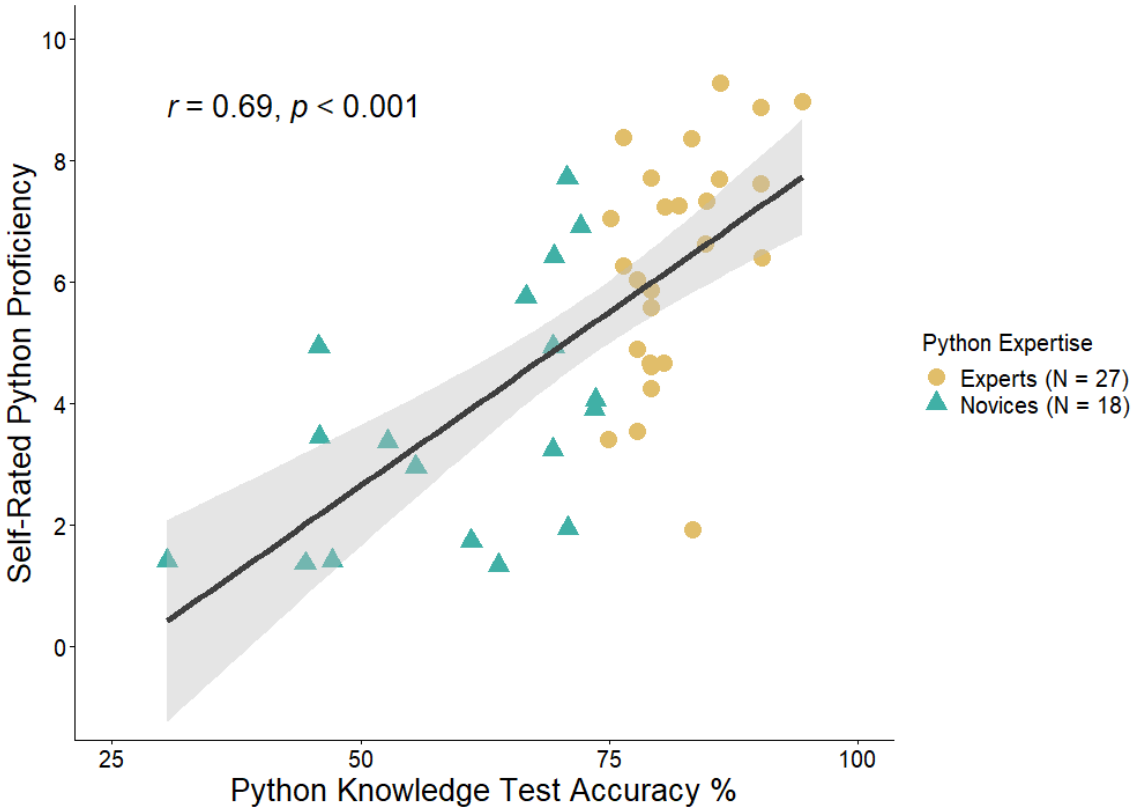

Fig. 2. Correlation between accuracy on the Python knowledge test (0-100%) and average self-rated Python proficiency (0-10).

3.2. English Proficiency Descriptives

Descriptive statistics for all English-related experience and proficiency measures are reported in Table 3. Performance on the NDRT yielded 20 English-High individuals and 21 English-Moderate individuals (Fig. 3). Four individuals who scored equal to the median (86.84) were excluded from either subgroup. Accuracy on the NDRT correlated with participants' overall self-rated English proficiency, $r(43) = .39$, $p = .008$ (Fig. 4).

Due to the nature of recruitment, there was an imbalance in the number of Python Experts vs Novices in the two English reading comprehension ability groups (English-High: 14 Experts, 6 Novices; English-Moderate: 11 Experts, 10 Novices). Two checks were used to ensure that the imbalance would not introduce a confound in subsequent analyses. First, a Welch's t-test indicated no significant difference in Python proficiency between the English-High and English-

Moderate groups, $t(38.94) = 1.53$, $p = .135$. Second, we conducted an analysis of a subset of the participants in which Python proficiency was matched across English reading comprehension ability, with 7 Experts and 5 Novices of nearly identical proficiency levels in each English proficiency group. Inspection of the ERP waveforms for these matched subgroups did not look qualitatively different from the larger groups. Therefore, the sample of 41 individuals (20 English-High, 21 English-Moderate) were used in analyses involving English reading comprehension ability in order to preserve power.

Table 3.
Descriptive statistics for English-related experience and proficiency measures.

English Measure	Group	Mean	SD	Min	Max
Age of first exposure to English	Whole Group	2.07	2.70	0.00	9.00
	High	0.75	1.12	0.00	4.00
	Moderate	3.57	3.20	0.00	9.00
Age beginning to read in English	Whole Group	5.64	2.64	0.00	13.00
	High	4.70	1.95	0.00	10.00
	Moderate	6.48	2.71	3.00	13.00
Age achieving general fluency in English	Whole Group	7.02	5.80	0.00	24.00
	High	4.95	3.52	0.00	15.00
	Moderate	8.95	6.88	0.00	24.00
Age achieving reading fluency in English	Whole Group	9.33	5.08	0.00	22.00
	High	7.75	4.28	1.00	18.00

	Moderate	10.95	5.58	4.00	22.00
Including English, number of languages with familiarity (1-3)	Whole Group	2.09	0.79	1.00	3.00
	High	1.90	0.79	1.00	3.00
	Moderate	2.29	0.78	1.00	3.00
Overall self-rated English proficiency (0-10)	Whole Group	9.30	1.09	6.33	10.00
	High	9.57	0.90	7.00	10.00
	Moderate	9.03	1.23	6.33	10.00
Reading proficiency	Whole Group	9.13	1.60	3.00	10.00
	High	9.45	1.36	6.00	10.00
	Moderate	8.81	1.83	3.00	10.00
Speaking proficiency	Whole Group	9.27	1.18	6.00	10.00
	High	9.50	1.00	6.00	10.00
	Moderate	9.00	1.34	6.00	10.00
Understanding proficiency	Whole Group	9.49	0.92	6.00	10.00
	High	9.75	0.55	8.00	10.00
	Moderate	9.29	1.15	6.00	10.00
Daily % reading English (0-100%)	Whole Group	91.53	14.38	50.00	100.00
	High	98.25	4.57	80.00	100.00
	Moderate	84.71	18.07	50.00	100.00

Daily % speaking English (0-100%)	Whole Group	82.84	24.54	10.00	100.00
	High	89.45	23.14	10.00	100.00
	Moderate	74.86	25.89	30.00	100.00
Daily % listening to English (0-100%)	Whole Group	85.04	20.69	35.00	100.00
	High	93.85	10.81	60.00	100.00
	Moderate	75.29	24.95	35.00	100.00
Nelson-Denny reading comprehension test (0-100%)	Whole Group	81.81	17.15	26.32	100.00
	High	94.87	3.13	89.47	100.00
	Moderate	68.42	16.54	26.32	84.21

Note. Sample sizes: Whole Group ($N = 45$); English-High ($N = 20$); English-Moderate ($N = 21$). Four individuals who scored equal to the median (86.84) on the Nelson-Denny Reading Test were excluded from the English-High and English-Moderate subgroups. Two participants reported not having achieved reading fluency in English; thus, the sample sizes for this measure were: Whole Group ($N = 43$); English-High ($N = 20$); English-Moderate ($N = 19$).

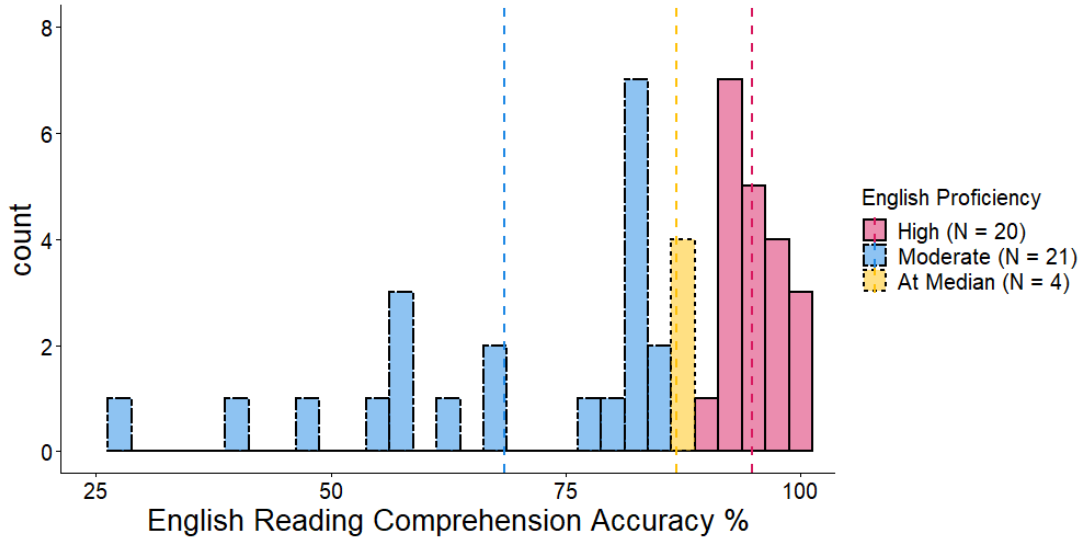

Fig. 3. Distributions of English reading comprehension ability, as measured by accuracy on the Nelson-Denny Reading Test. Individuals who scored equal to the whole-group median (86.84) are included for complete representation of the entire sample. Dashed vertical lines denote the means per sub-group.

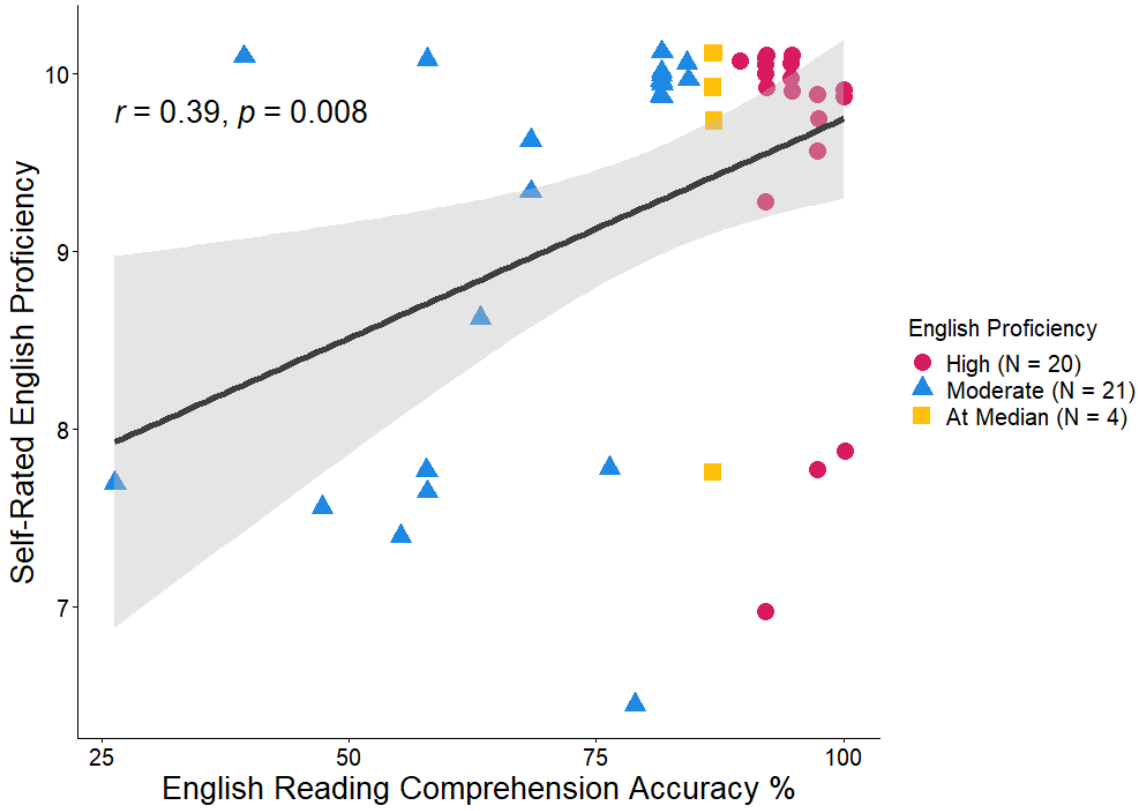

Fig. 4. Correlation between accuracy on the Nelson-Denny Reading Test (0-100%) and average self-rated English proficiency (0-10).

3.3. Whole Group Analyses

3.3.1. Code acceptability judgment task. Average acceptability judgment rates per condition for all participants are provided in Table 4. Overall, there was a main effect of semantic plausibility, $F(1, 44) = 16.51$, $p < .001$, and main effect of structural validity, $F(1, 44) = 110.43$, $p < .001$, but no interaction effect. Holm-Bonferroni adjusted post-hoc tests revealed that acceptability rates were different for all condition pairs (Fig. 5). Participants judged well-formed code to be more acceptable than semantically implausible code, $t(44) = 3.14$, $p = .003$, structurally invalid code, $t(44) = 10.97$, $p < .001$, and doubly anomalous code, $t(44) = 15.52$, $p < .001$. Semantically implausible code was judged to be more acceptable than structurally invalid code and doubly anomalous code, while structurally invalid code was judged to be more acceptable than doubly anomalous code, ps $< .001$.

Table 4.
Average acceptability judgments for all participants ($N = 45$).

	Mean	**SD**	**Min**	**Max**
Well-formed	77.22	21.17	10.00	100.00
Semantically implausible	64.63	30.43	5.00	100.00
Structurally invalid	25.61	24.33	0.00	82.50
Doubly anomalous	14.89	15.10	0.00	55.00

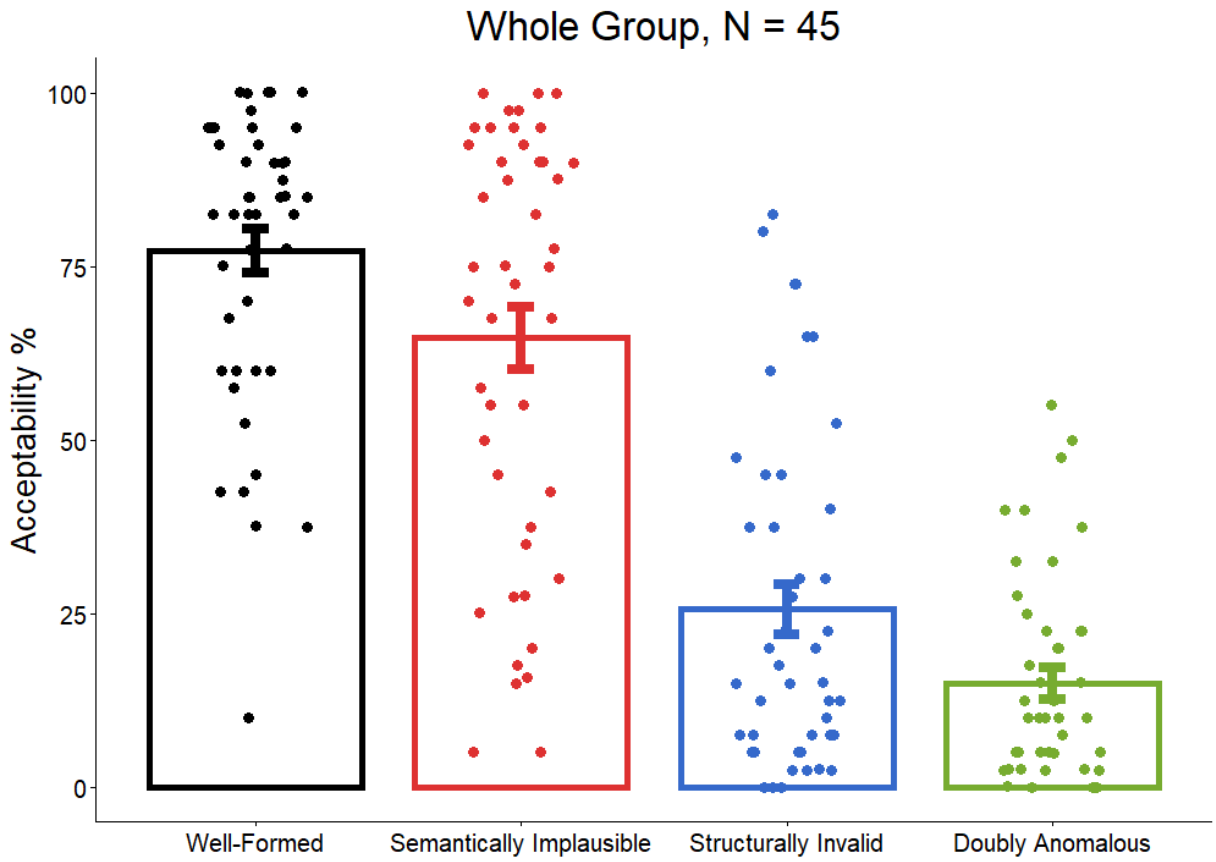

Fig. 5. Acceptability judgments for all participants. Error bars represent one standard error of the mean.

3.3.2. Grand mean ERPs. Whole group ERPs for semantic plausibility (Fig. 6A), structural validity (Fig. 6B), and double anomalies (Fig. 6C) are plotted separately. Plots showing change in amplitude between the well-formed condition and each anomalous condition at the Cz electrode can be found in Fig. 7.

3.3.2.1. N400 (300-500 ms) time window. Visual inspection of the grand mean ERP waveforms indicated that relative to well-formed code, semantically implausible code (Fig. 6A) and doubly anomalous code (Fig. 6C) elicited a negativity between 300 and 500 ms along the midline. In line with these observations, statistical analyses revealed a main effect of semantic plausibility, $F(1, 44) = 26.93$, $p < .001$, with the effect being stronger towards the posterior, semantic plausibility x electrode interaction: $F(1.41, 62.08) = 10.77$, $p < .001$. There was also a

main effect of structural validity, $F(1, 44) = 26.76$, $p < .001$, that varied by electrode, with visual inspection suggesting a strong positivity at the Fz electrode pulling the waveform for structurally invalid code below the well-formed code, structural validity x electrode interaction: $F(1.35, 58.28) = 11.36$, $p < .001$. However, there was no significant interaction between semantic plausibility and structural validity, nor did this effect vary by electrode. Overall, these results suggest that within the 300-500 ms time window, participants exhibited N400 responses to semantically implausible code and doubly anomalous code, but not structurally invalid code. The effect of semantic plausibility is exhibited similarly regardless of the structural validity of the code.

3.3.2.2. P600 (500-800 ms) time window. Visual inspection of grand mean ERP waveforms indicated that relative to well-formed code, structurally invalid code (Fig. 6B) and doubly anomalous code (Fig. 6C) elicited a positivity beginning around 500 ms along the midline. In line with these observations, statistical analyses revealed a main effect of structural validity, $F(1, 44) = 70.29$, $p < .001$. There was no main effect of semantic plausibility or interaction between semantic plausibility and structural validity. These results suggest that within the 500-800 ms time window, participants exhibited P600 effects to structurally invalid and doubly anomalous code, but not semantically implausible code. The effect of structural validity is exhibited similarly regardless of the semantic plausibility of the code.

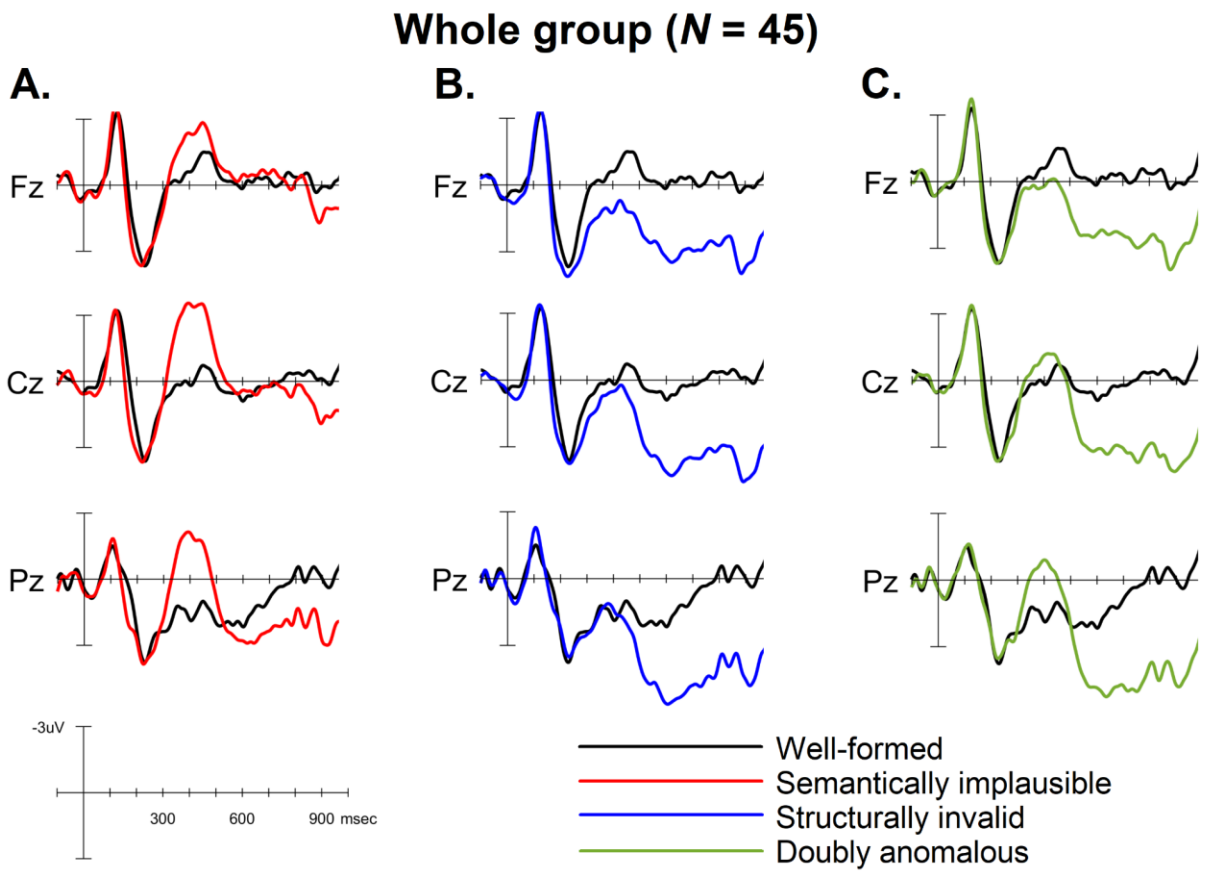

Fig. 6. Grand mean ERP waveforms for all participants ($N = 45$) comparing well-formed code to A) semantically implausible code, B) structurally invalid code, and C) doubly anomalous code. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

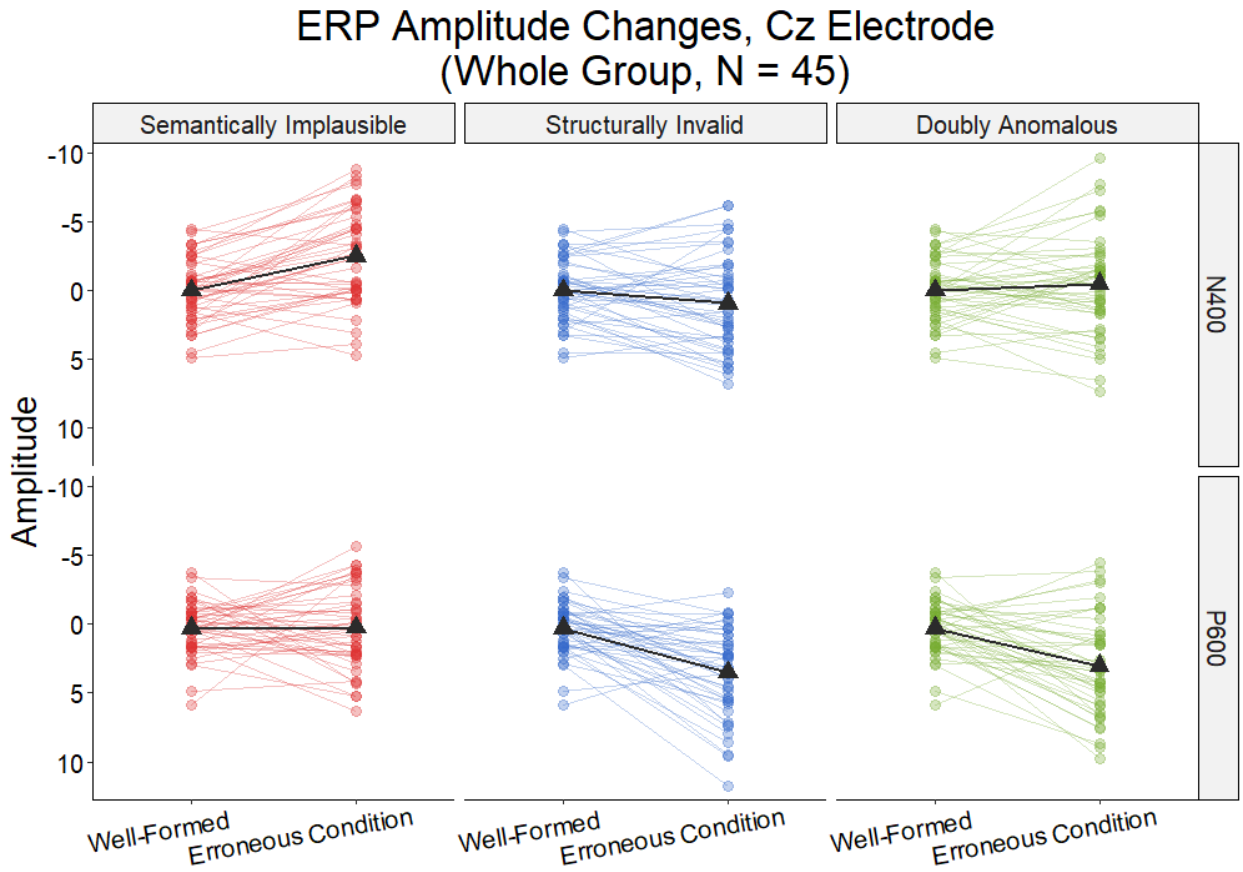

Fig. 7. Change in average ERP amplitude at the Cz electrode site for all participants ($N = 45$) from well-formed code to semantically implausible code (left, red), structurally invalid code (center, blue), and doubly anomalous code (right, green). Top row: average amplitude in the N400 window (300-500 ms); bottom row: average amplitude in the P600 window (500-800 ms). Triangle denotes the mean. Negative voltage is plotted up.

3.4. Between-Group Analyses: Python Experts versus Python Novices

3.4.1. Code acceptability judgment task. Average acceptability judgment rates per condition for Python Experts and Python Novices are provided in Table 5. In addition to the previously reported whole-group main effects of semantic plausibility and structural validity, there was an interaction between structural validity and expertise, $F(1, 43) = 25.62$, $p < .001$. The interaction between semantic plausibility and expertise approached significance, $F(1, 43) = 3.42$, $p = .071$, whereas the three-way interaction between semantic plausibility, structural validity, and expertise was not significant, $F(1, 43) = 0.33$, $p = .955$. Holm-Bonferroni adjusted post-hoc tests

revealed that Experts judged structurally invalid code and doubly anomalous code to be less acceptable compared to Novices, $t(43) = -2.01$, $p < .001$ and $t(43) = -4.37$, $p < .001$ respectively. Experts judged semantically implausible code to be more acceptable compared to Novices, but this difference did not survive corrections for multiple comparisons, $t(43) = 2.68$, $p = .073$ ($p = .010$ uncorrected). Experts and Novices did not differ in their acceptability judgment rates for well-formed code, $t(43) = 2.01$, $p = .220$.

Follow-up analyses with Experts and Novices in separate ANOVA models (Bonferroni adjusted family-wise alpha = .025 each) revealed differences in acceptability rates within each Python expertise group. In Experts, there was a main effect of structural validity, $F(1, 26) = 187.08$, $p < .001$, but no main effect of semantic plausibility, $F(1, 26) = 3.76$, $p = .063$, or interaction between semantic plausibility and structural validity, $F(1, 26) = 0.24$, $p = .628$. Holm-Bonferroni adjusted post-hoc tests revealed that Experts judged well-formed code to be more acceptable than structurally invalid code, $t(26) = 14.49$, $p < .001$, and doubly anomalous code, $t(26) = 17.18$, $p < .001$, but not semantically implausible code, $t(26) = 1.57$, $p = .128$. Additionally, Experts judged semantically implausible code to be more acceptable than structurally invalid code, $t(26) = 7.75$, $p < .001$, and doubly anomalous code, $t(26) = 11.62$, $p < .001$. However, Experts judged structurally invalid code to be similarly acceptable as doubly anomalous code, $t(26) = 2.36$, $p = .052$. These results suggest that Experts focus on structural validity to judge the acceptability of code, regardless of semantic plausibility.

In Novices, there were main effects of both semantic plausibility, $F(1, 17) = 20.72$, $p < .001$, and structural validity, $F(1, 17) = 17.94$, $p < .001$, but no interaction, $F(1, 17) = 0.15$, $p = .701$. Unlike Experts, Novices judged well-formed code to be more acceptable than semantically implausible code, $t(17) = 3.17$, $p = .011$, structurally invalid code, $t(17) = 4.52$, p

= .001, and doubly anomalous code, $t(17) = 7.99$, $p < .001$, all Holm-Bonferroni adjusted. Furthermore, Novices judged doubly anomalous code to be less acceptable than semantically implausible code, $t(17) = -3.43$, $p = .010$, and structurally invalid code, $t(17) = -5.52$, $p < .001$, but judged semantically implausible code and structurally invalid code to be similarly acceptable, $t(17) = 0.98$, $p = .340$. These results suggest that while Novices are able to distinguish non-anomalous code from anomalous code, they are similarly affected in their acceptability judgments when only one violation is present, regardless of type. It is only when double anomalies are present do Novices perform closer to Experts.

Table 5.
Average acceptability judgments for Python Experts ($N = 27$) and Python Novices ($N = 18$).

	Mean	**SD**	**Min**	**Max**
	Python Experts (N = 27)			
Well-formed	82.22	19.23	37.50	100.00
Semantically implausible	73.92	27.66	5.00	100.00
Structurally invalid	14.72	17.60	0.00	65.00
Doubly anomalous	8.15	10.27	0.00	40.00
	Python Novices (N = 18)			
Well-formed	69.72	22.24	10.00	95.00
Semantically implausible	50.69	29.74	5.00	95.00
Structurally invalid	41.94	24.25	5.00	82.50
Doubly anomalous	25.00	15.76	0.00	55.00

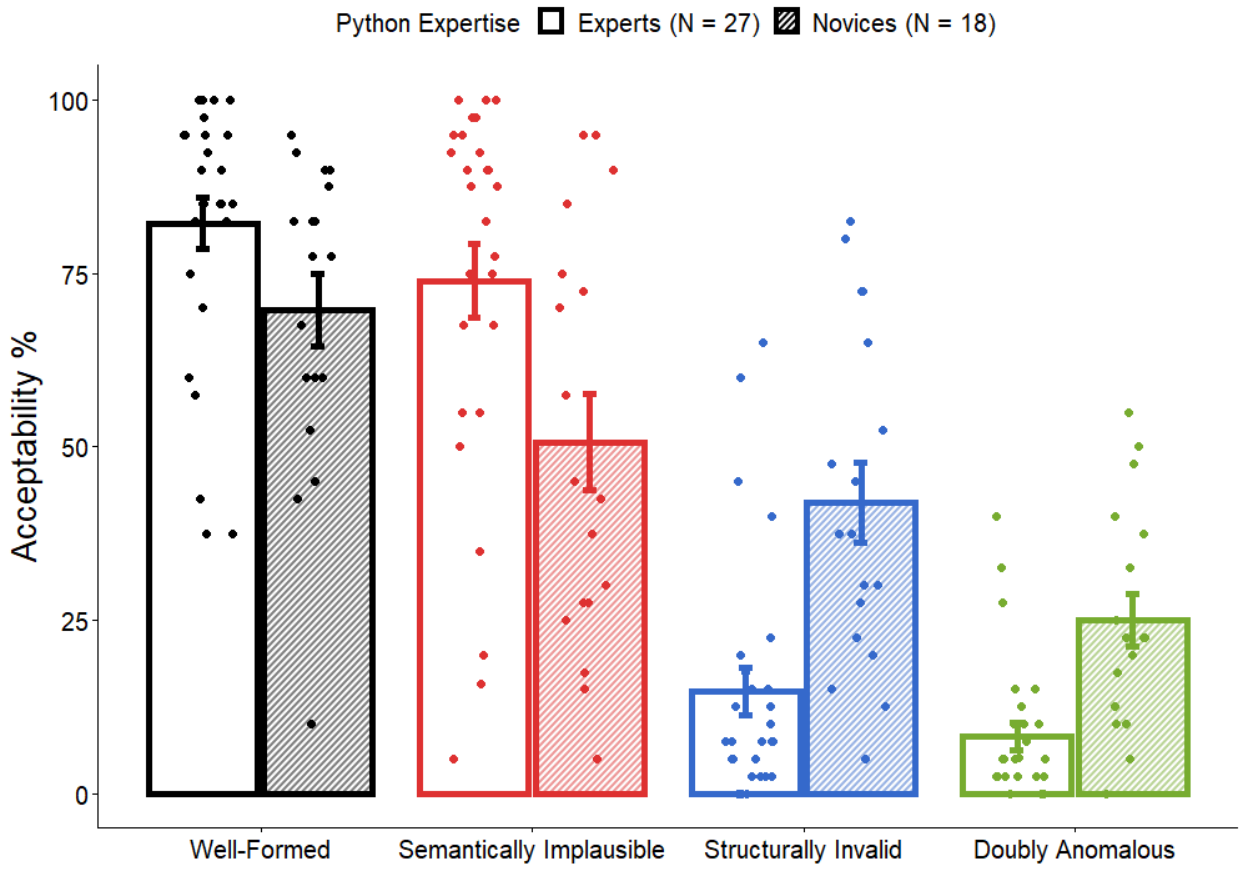

Fig. 8. Acceptability judgments split by Python expertise. Error bars represent one standard error of the mean.

3.4.2. Grand mean ERPs. ERPs for semantic plausibility, structural validity, and double anomalies are plotted separately for Python Experts (Fig. 9) and Python Novices (Fig. 11). Plots showing change in amplitude between the well-formed condition and each anomalous condition at the Cz electrode can be found in Fig. 10 (Experts) and Fig. 12 (Novices). A summary of the ERP differences between Experts and Novices can be found in Fig. 13.

3.4.2.1. N400 (300-500 ms) time window. Overall, Experts and Novices did not differ in any of the analyses within the N400 effect window. However, the interaction between structural validity and expertise approached significance, $F(1, 43) = 3.83$, $p = .057$, as did the three-way interaction between semantic plausibility, structural validity, and expertise, $F(1, 43) = 3.83$, $p = .057$. Visual inspection suggests that this is driven by a stronger positivity at the more anterior

electrodes pulling the waveform for structurally invalid code below the well-formed code for Experts.

3.4.2.2. P600 (500-800 ms) time window. Compared to Novices, Experts exhibited a stronger positivity to structurally invalid and doubly anomalous code than to well-formed code within the P600 effect window, which was supported by statistical analyses: structural validity by expertise interaction, $F(1, 43) = 7.01$, $p = .011$. Follow-up analyses with Experts and Novices in separate ANOVA models (Bonferroni adjusted family-wise alpha = .025 each) revealed that both groups exhibited P600 effects to structurally invalid and doubly anomalous code relative to well-formed code: Experts, $F(1, 26) = 63.97$, $p < .001$; Novices, $F(1, 17) = 17.03$, $p = .001$. Experts and Novices did not differ in any other patterns in the P600 window.

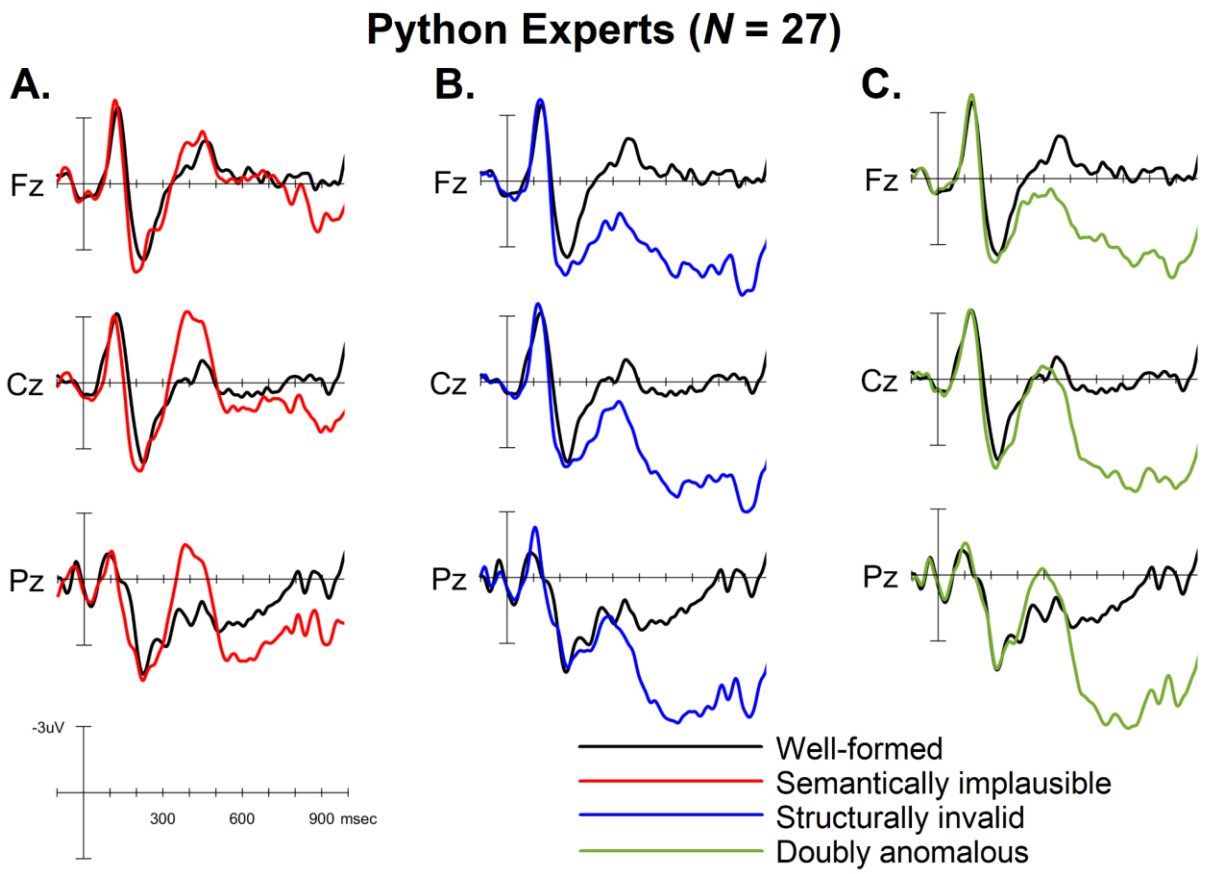

Fig. 9. Grand mean ERP waveforms for Python Experts ($N = 27$) comparing well-formed code to A) semantically implausible code, B) structurally invalid code, and C) doubly anomalous code. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

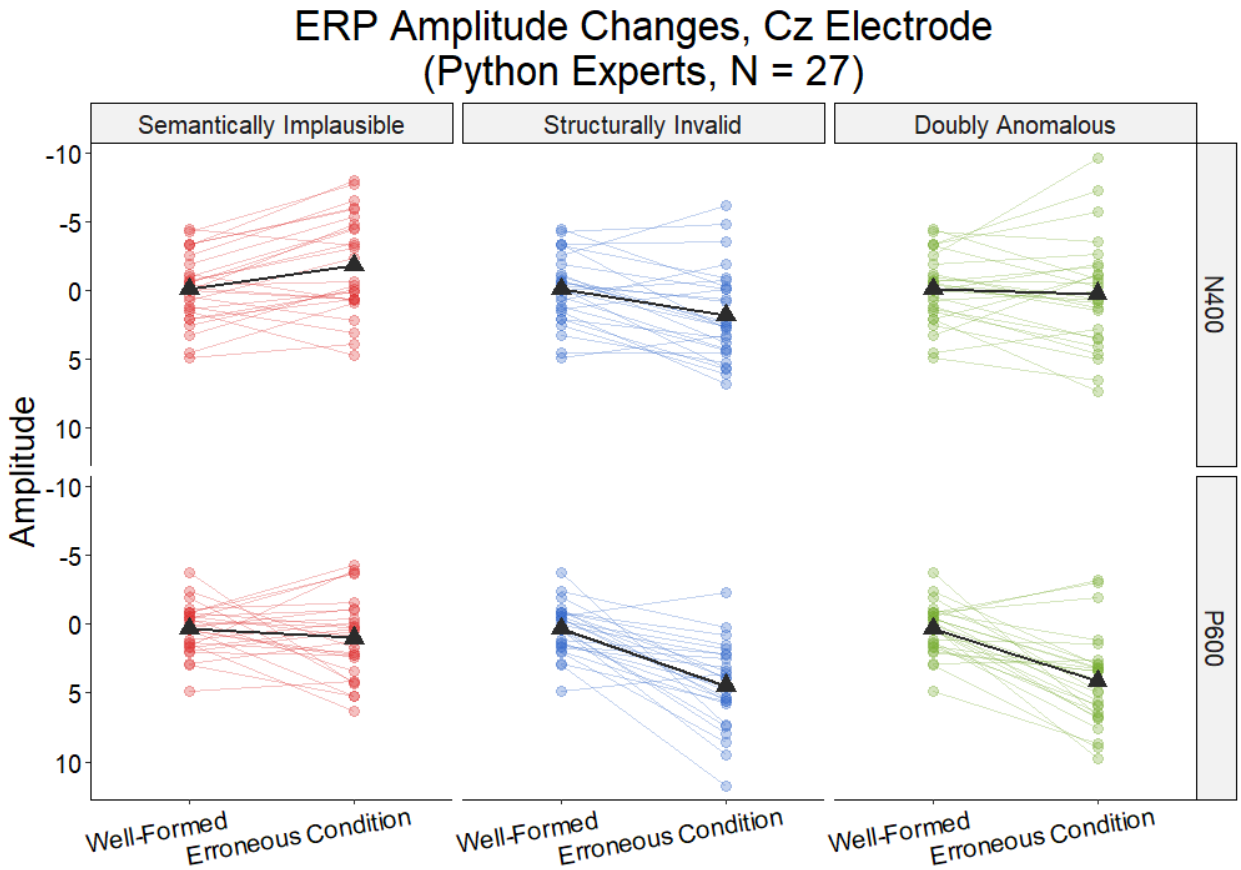

Fig. 10. Change in average ERP amplitude at the Cz electrode site for Python Experts ($N = 27$) from well-formed code to semantically implausible code (left, red), structurally invalid code (center, blue), and doubly anomalous code (right, green). Top row: average amplitude in the N400 window (300-500 ms); bottom row: average amplitude in the P600 window (500-800 ms). Triangle denotes the mean. Negative voltage is plotted up.

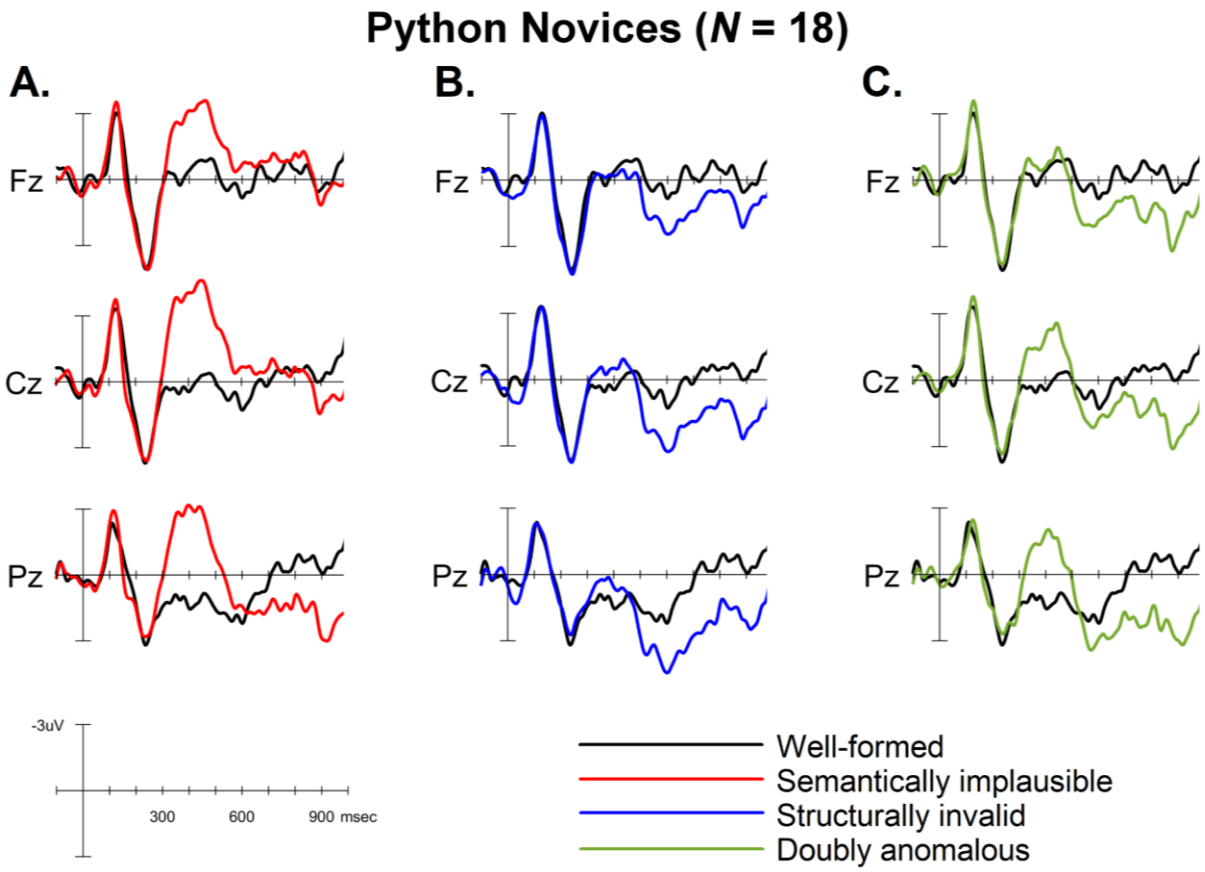

Fig. 11. Grand mean ERP waveforms for Python Novices (N = 18) comparing well-formed code to A) semantically implausible code, B) structurally invalid code, and C) doubly anomalous code. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

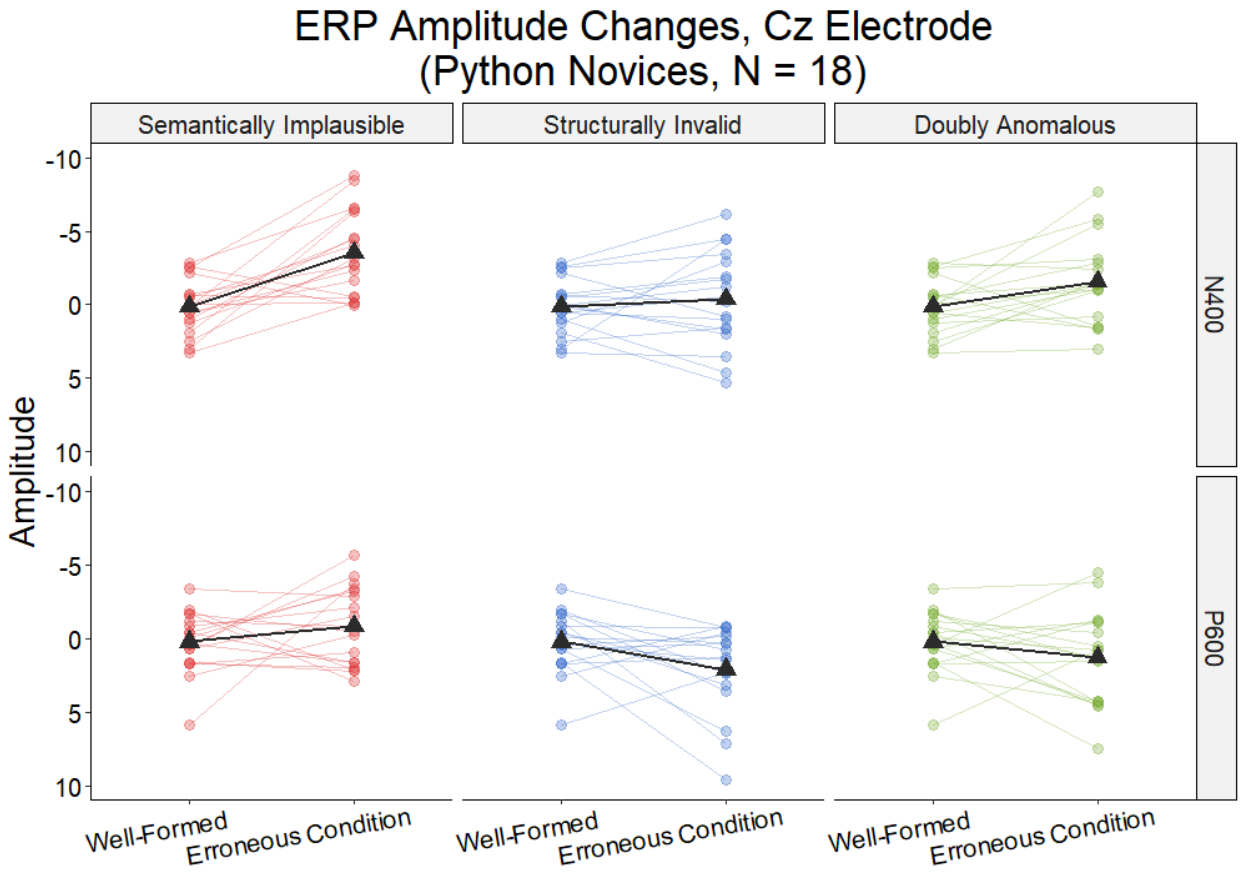

Fig. 12. Change in average ERP amplitude at the Cz electrode site for Python Novices ($N = 18$) from well-formed code to semantically implausible code (left, red), structurally invalid code (center, blue), and doubly anomalous code (right, green). Top row: average amplitude in the N400 window (300-500 ms); bottom row: average amplitude in the P600 window (500-800 ms). Triangle denotes the mean. Negative voltage is plotted up.

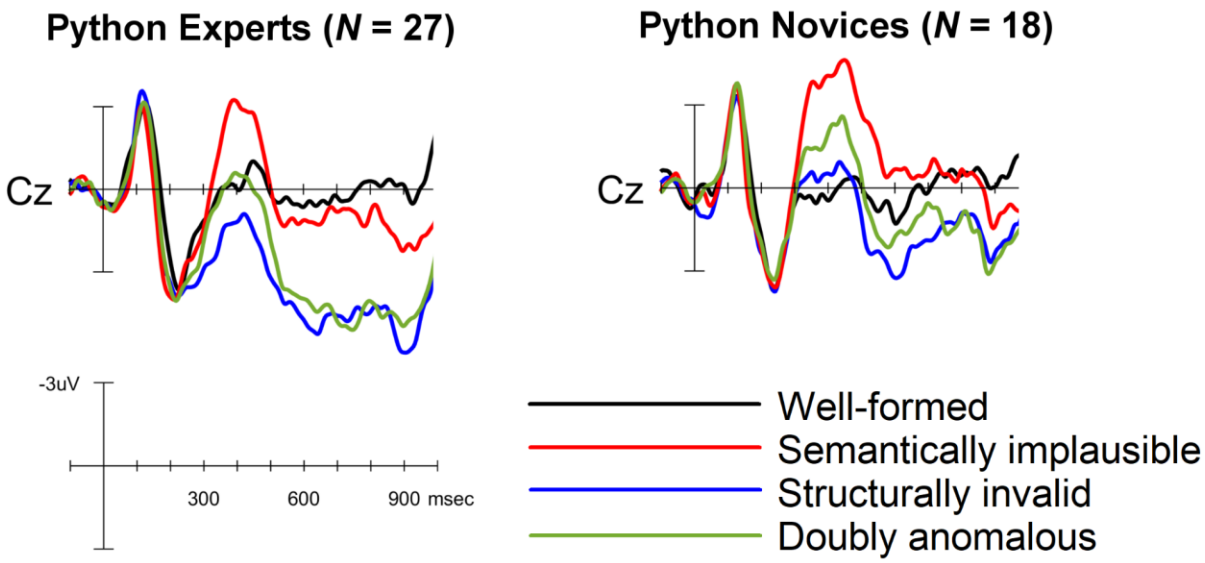

Fig. 13. Summary of the grand mean ERP waveforms for Python Experts (N = 27) versus Python Novices (N = 18) for all four conditions at the Cz electrode. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

3.5. Between-Group Analyses: English-High versus English-Moderate Individuals

3.5.1. Acceptability judgment task. Average acceptability judgment rates per condition for English-High individuals and English-Moderate individuals are provided in Table 6. In addition to the previously reported whole-group main effects of semantic plausibility and structural validity, there was an interaction between structural validity and English reading comprehension ability, $F(1, 39) = 4.48$, $p = .041$. Post-hoc tests revealed that English-High individuals judged structurally invalid code and doubly anomalous code to be less acceptable compared to English-Moderate individuals, but these differences did not survive Holm-Bonferroni adjustments for multiple comparisons, $t(39) = -2.73$, $p = .092$ ($p = .009$ uncorrected) and $t(39) = -2.74$, $p = .092$ ($p = .009$ uncorrected respectively. English-High individuals and English-Moderate individuals did not differ in their acceptability rates for well-formed code, $t(39) = -0.17$, $p = 1.000$, or semantically implausible code, $t(39) = 1.19$, $p = 1.000$.

Follow-up analyses with English-High individuals and English-Moderate individuals in separate ANOVA models (Bonferroni adjusted family-wise alpha = .025 each) revealed differences in acceptability rates within each English reading comprehension ability group. In English-High individuals, there was a main effect of structural validity, $F(1, 19) = 59.09$, $p < .001$, but no main effect of semantic plausibility, $F(1, 19) = 3.53$, $p = .076$, or interaction between semantic plausibility and structural validity, $F(1, 19) < 0.01$, $p = .962$. Holm-Bonferroni adjusted post-hoc tests revealed that English-High individuals judged well-formed code to be more acceptable than structurally invalid code, $t(19) = 7.96$, $p < .001$, and doubly anomalous code, $t(19) = 10.02$, $p < .001$, but not semantically implausible code, $t(19) = 1.25$, $p = .226$. Additionally, English-High individuals judged semantically implausible code to be more acceptable than structurally invalid code, $t(19) = 5.17$, $p < .001$, and doubly anomalous code, $t(19) = 6.91$, $p < .001$, while structurally invalid code was judged to be more acceptable than doubly anomalous code, $t(19) = 2.88$, $p = .019$. These results suggest that English-High individuals find structural anomalies to be more problematic than semantic anomalies, and that having both anomalies is the most egregious scenario.

In English-Moderate individuals, there were main effects of both semantic plausibility, $F(1, 20) = 12.35$, $p = .002$, and structural validity, $F(1, 20) = 45.24$, $p < .001$, but no interaction, $F(1, 20) = 1.38$, $p = .255$. Unlike English-High individuals, English-Moderate individuals judged well-formed code to be more acceptable than semantically implausible code, $t(20) = 3.04$, $p = .013$, structurally invalid code, $t(20) = 7.081$, $p < .001$, and doubly anomalous code, $t(20) = 10.87$, $p < .001$, all Holm-Bonferroni adjusted. Furthermore, English-Moderate individuals judged doubly anomalous code to be less acceptable than semantically implausible code, $t(20) = -5.54$, $p < .001$, and structurally invalid code, $t(20) = -3.64$, $p = .005$, but judged semantically

implausible code and structurally invalid code to be similarly acceptable, $t(20) = 2.37$, $p = .028$. Overall, the differences in acceptability judgment rates within English-Moderate individuals mirror the differences found in Python Novices.

Table 6.
Average acceptability judgments for English-High individuals ($N = 20$) and English-Moderate individuals ($N = 21$).

	Mean	**SD**	**Min**	**Max**
	English-High (N = 20)			
Well-formed	75.38	25.82	10.00	100.00
Semantically implausible	68.29	31.62	5.00	100.00
Structurally invalid	15.63	20.77	0.00	82.50
Doubly anomalous	8.76	13.21	0.00	55.00
	English-Moderate (N = 21)			
Well-formed	76.55	17.65	42.50	100.00
Semantically implausible	56.90	29.88	5.00	100.00
Structurally invalid	34.64	23.65	2.50	80.00
Doubly anomalous	20.36	13.84	0.00	47.50

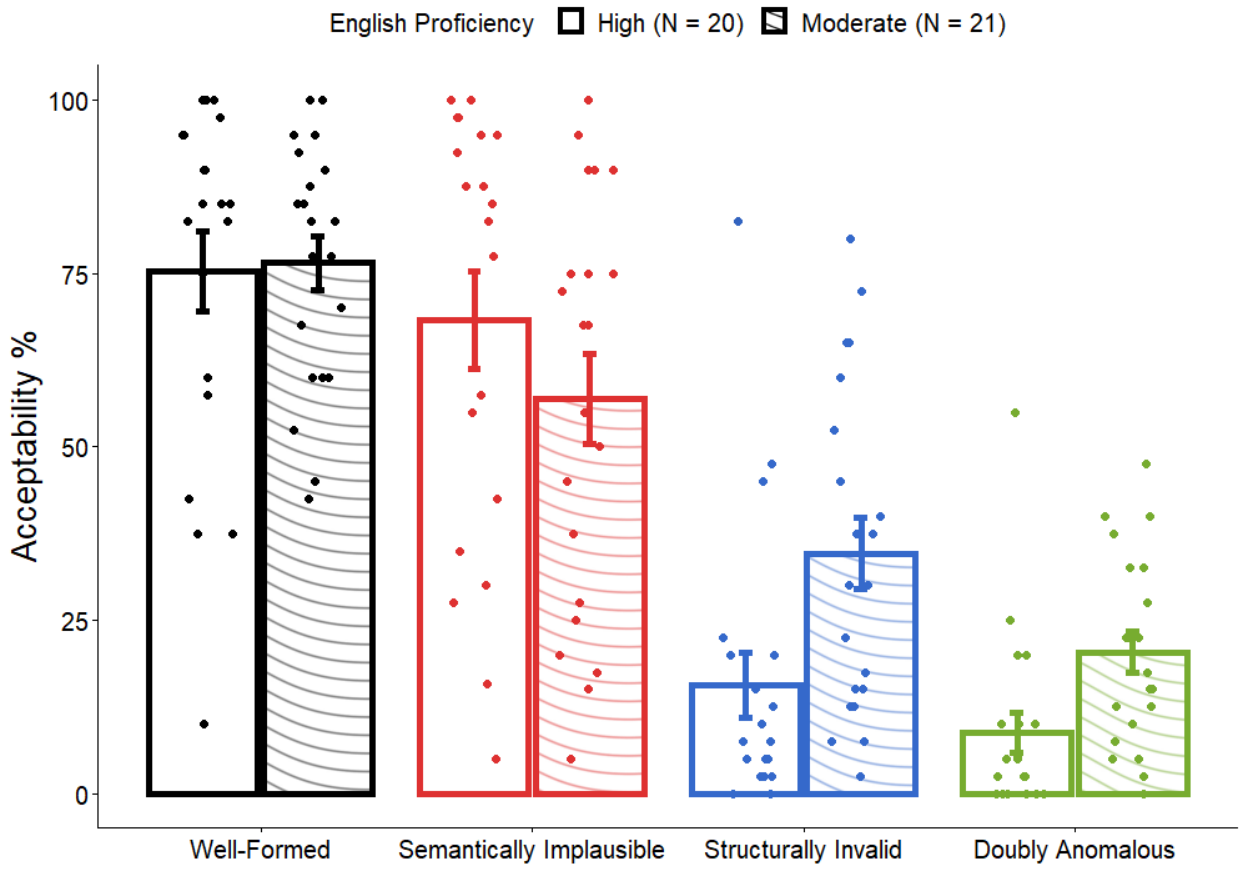

Fig. 14. Acceptability judgments split by English proficiency. Error bars represent one standard error of the mean.

3.5.2. Grand mean ERPs. ERPs for semantic plausibility, structural validity, and double anomalies are plotted separately for English-High individuals (Fig. 15) and English-Moderate individuals (Fig. 17) . Plots showing change in amplitude between the well-formed condition and each anomalous condition at the Cz electrode can be found in Fig. 16 (English-High) and Fig. 18 (English-Moderate). A summary of the ERP differences between English-High individuals and English-Moderate individuals can be found in Fig. 19.

3.5.2.1. N400 (300-500 ms) time window. Statistical analyses on the $N = 41$ group revealed similar whole group effects as the $N = 45$ group in the 300-500 ms time window. Specifically, there was a main effect of semantic plausibility, which varied by electrode. There was also a main effect of structural validity that resulted from the positivity at the Fz electrode

pulling the waveform for structurally invalid code below the well-formed code. No interaction effect between semantic plausibility and structural validity was found. None of these effects had significant interactions with English reading comprehension ability, suggesting that the N400 response in the 300-500 ms window does not differ between English-High individuals and English-Moderate individuals.

3.5.2.2. P600 (500-800 ms) time window. Statistical analyses on the $N = 41$ group revealed similar whole group effects as the $N = 45$ group in the 500-800 ms time window. Specifically, there was a main effect of structural validity, but no main effect of semantic plausibility. Unlike the $N = 45$ group, however, there was no significant three-way interaction between semantic plausibility, structural validity, and electrode. None of these effects had significant interactions with English reading comprehension ability, suggesting that the P600 response in the 500-800 ms window does not differ between English-High individuals and English-Moderate individuals.

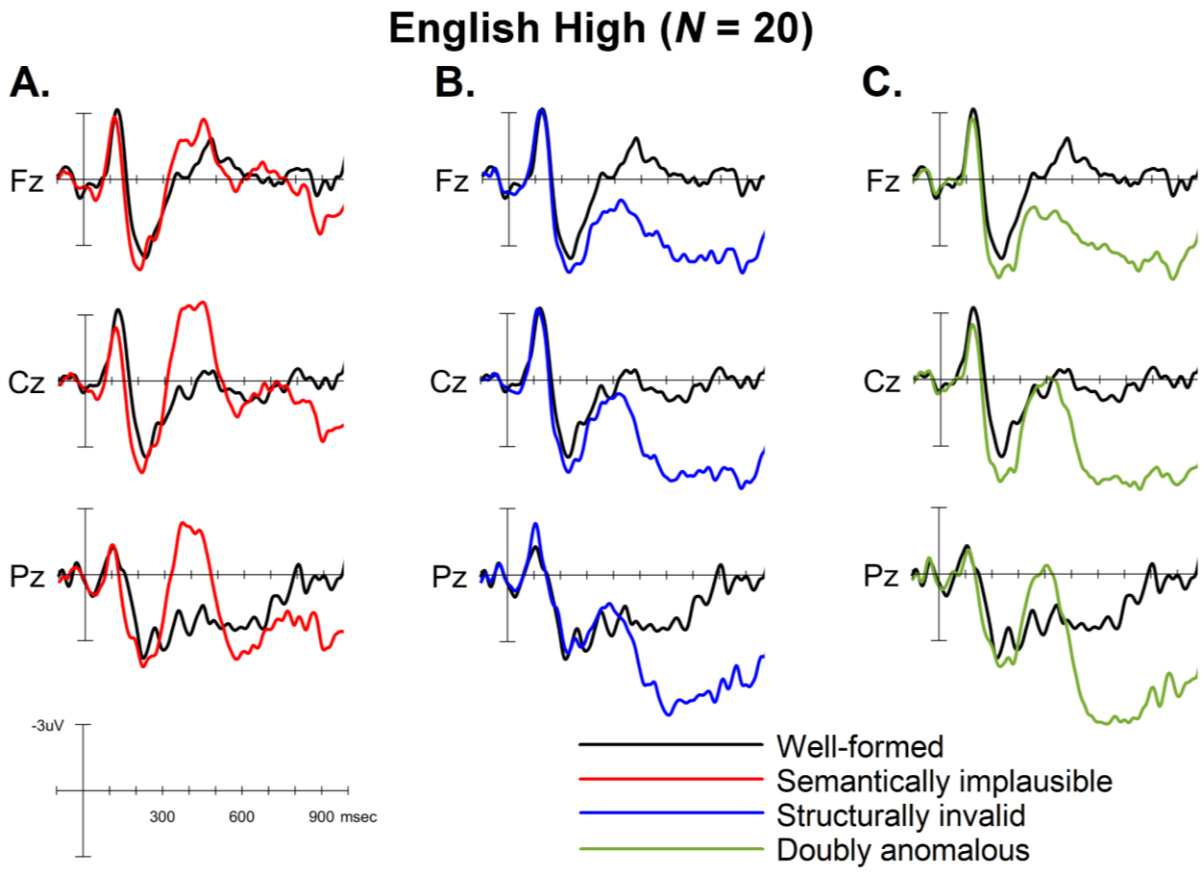

Fig. 15. Grand mean ERP waveforms for English-High individuals ($N = 20$) comparing well-formed code to A) semantically implausible code, B) structurally invalid code, and C) doubly anomalous code. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

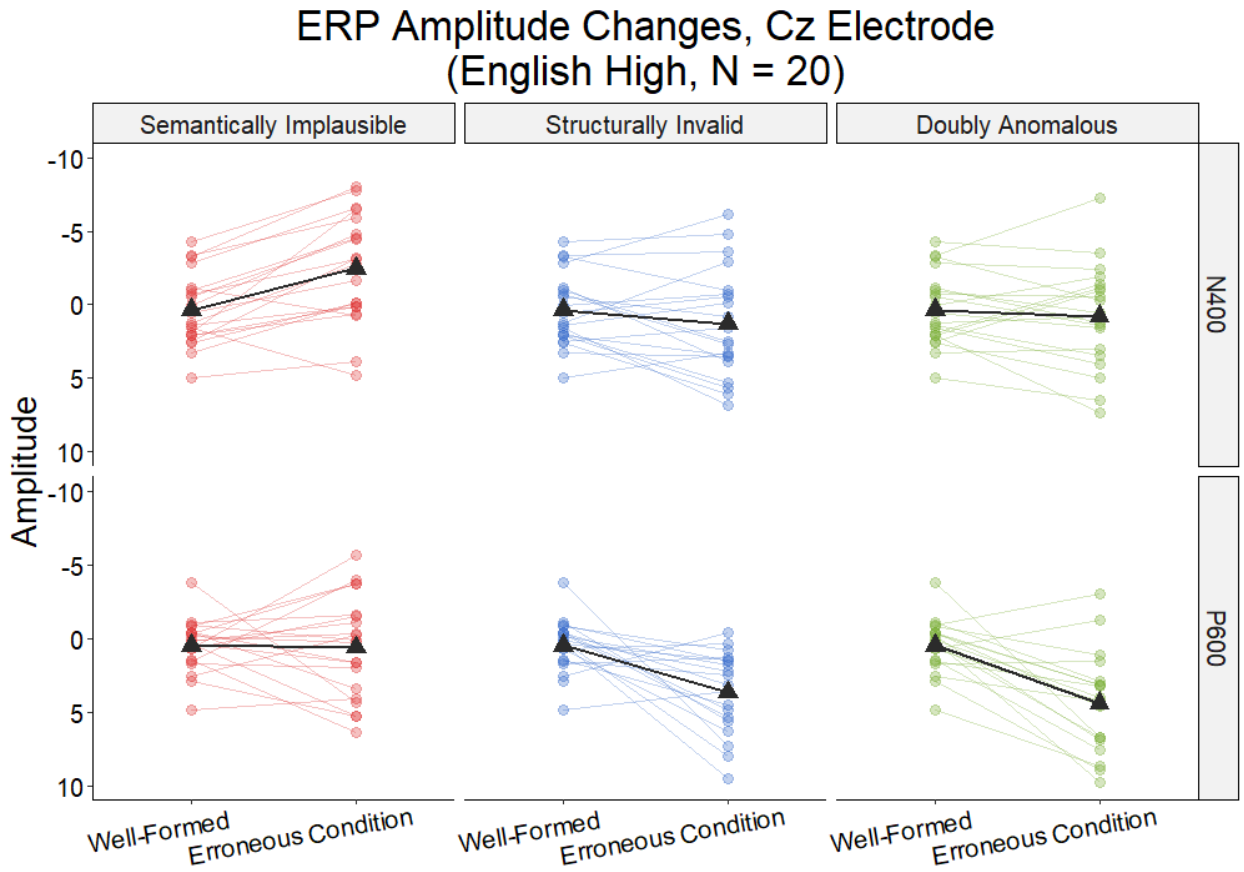

Fig. 16. Change in average ERP amplitude at the Cz electrode site for English-High individuals ($N = 20$) from well-formed code to semantically implausible code (left, red), structurally invalid code (center, blue), and doubly anomalous code (right, green). Top row: average amplitude in the N400 window (300-500 ms); bottom row: average amplitude in the P600 window (500-800 ms). Triangle denotes the mean. Negative voltage is plotted up.

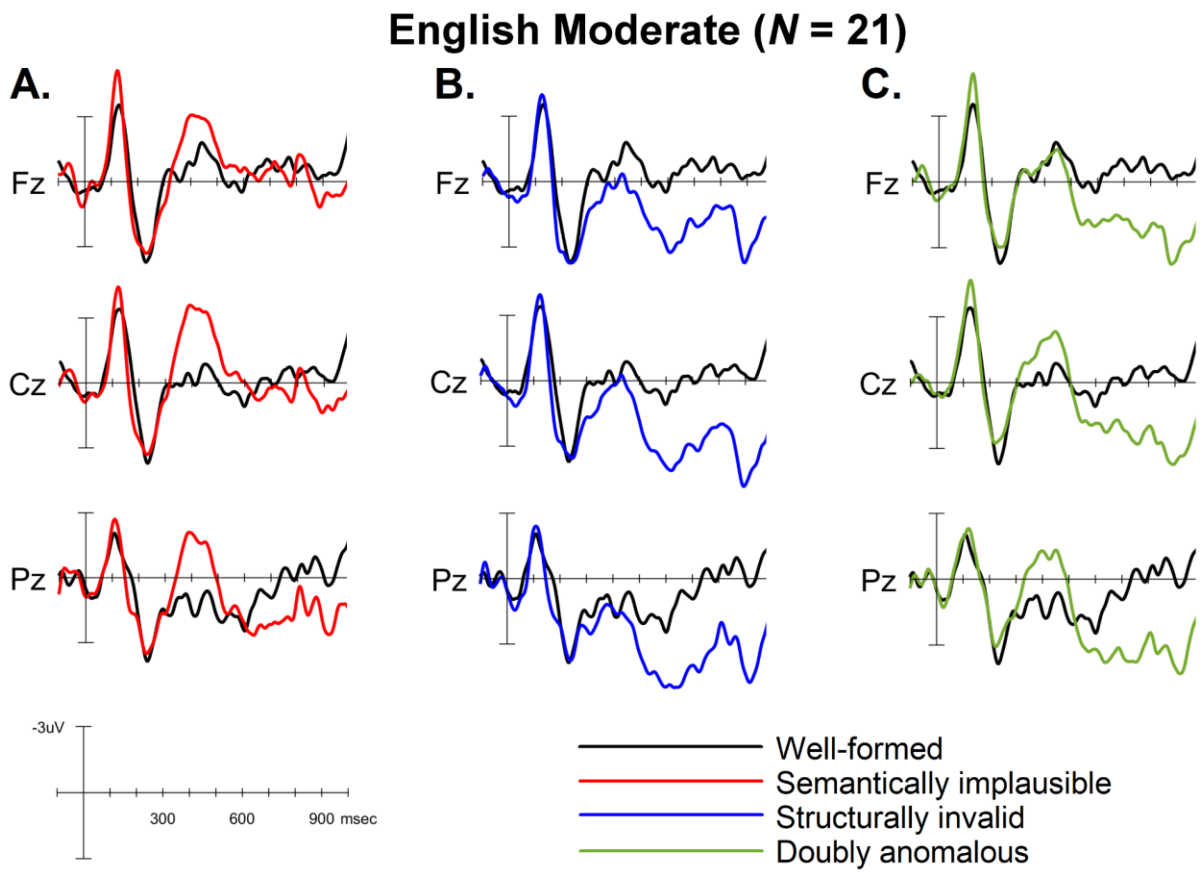

Fig. 17. Grand mean ERP waveforms for English-Moderate individuals ($N = 21$) comparing well-formed code to A) semantically implausible code, B) structurally invalid code, and C) doubly anomalous code. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

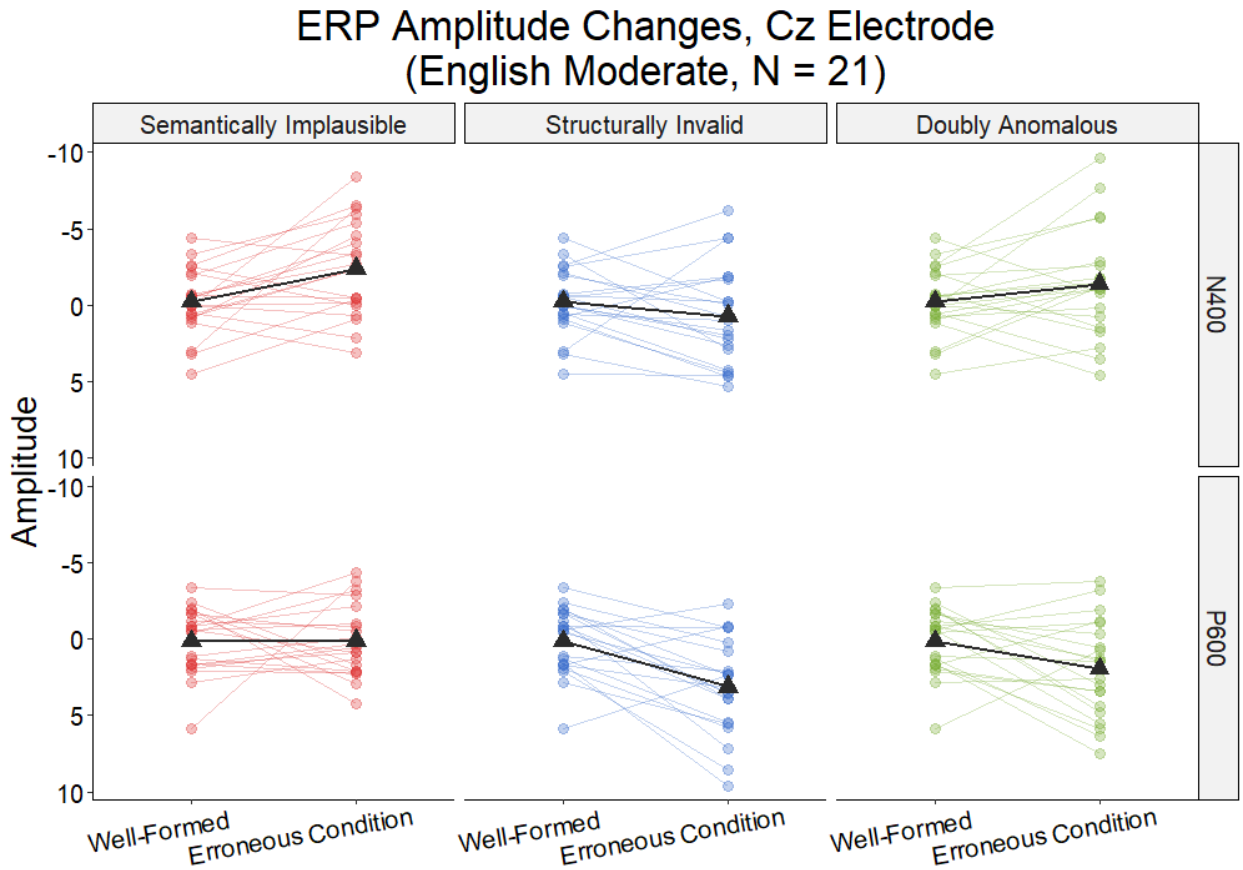

Fig. 18. Change in average ERP amplitude at the Cz electrode site for English-Moderate individuals ($N = 21$) from well-formed code to semantically implausible code (left, red), structurally invalid code (center, blue), and doubly anomalous code (right, green). Top row: average amplitude in the N400 window (300-500 ms); bottom row: average amplitude in the P600 window (500-800 ms). Triangle denotes the mean. Negative voltage is plotted up.

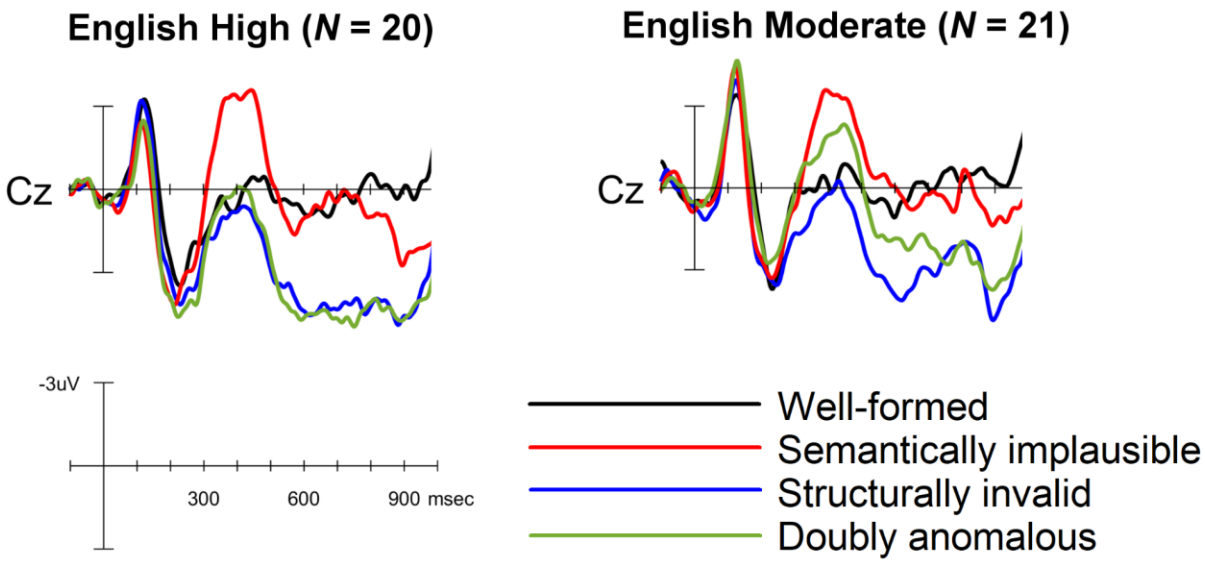

Fig. 19. Summary of the grand mean ERP waveforms for English-High individuals ($N = 20$) versus English-Moderate ($N = 21$) for all four conditions at the Cz electrode. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

3.6. Summary of ERP Results

Participants exhibited N400 effects to semantically implausible code and doubly anomalous code, as well as P600 effects to structurally invalid and doubly anomalous code. The N400 response to semantic plausibility did not differ as a result of group differences in Python expertise or English reading comprehension ability. However, the P600 response to structural validity was stronger in Python Experts than in Python Novices. Overall, these results suggest that Python programmers exhibit different brain responses to the semantic plausibility and structural validity of code, and responses to structure are presented differently in Experts and Novices.

On the other hand, none of the effects in the N400 window or the P600 window were different when participants were grouped by English reading comprehension ability. To explore whether this result may have been due to the dichotomization of reading comprehension ability,

bivariate correlations were run between Python proficiency, English reading comprehension ability, and four measures of mean difference amplitudes, all at the Cz electrode: 1) semantically implausible - well-formed (N400), 2) structurally invalid code - well-formed (P600), 3) doubly anomalous - well-formed (N400), and 4) doubly anomalous - well-formed (P600). As shown in Table 7, Python proficiency is positively correlated with the magnitude of P600 effects to structurally invalid code and doubly anomalous code, but not any N400 effects, reflective of the grand mean ERP results. On the other hand, English reading comprehension ability does not correlate with any of the mean difference amplitudes. However, controlling for English reading comprehension ability weakens the correlations between Python proficiency and all mean difference amplitudes, except for well-formed - semantically implausible (N400). As such,

Table 7.
Correlations of mean difference amplitudes (Cz electrode) with Python proficiency and English reading comprehension ability.

	Original *r* with Python proficiency	**Original *r* with English reading comprehension ability**	**Partial *r* with Python proficiency (controlled for English comprehension)**
Semantically implausible - well-formed (N400)	-.0001	-.21	.11
Structurally invalid - well-formed (P600)	.30*	.10	.28
Doubly anomalous - well-formed (N400)	.22	.11	.18
Doubly anomalous - well-formed (P600)	.32*	.22	.25

Note. *p <.05 (uncorrected). Partial correlations represent correlations between Python proficiency and various mean difference amplitudes, controlling for English reading comprehension ability. Bivariate correlation between Python proficiency and English reading comprehension ability is $r(43) = .46$, $p = .002$ (uncorrected).

English reading comprehension ability may contribute to differences in brain responses to anomalous code; more research will have to be done to determine the extent to which this is true.

3.7. Effect of Code Type (Variables versus Keywords)

Results for the whole group of 45 individuals are presented first, followed by the results for Python Experts vs Python Novices, concluding with English-High vs English-Moderate individuals.

3.7.1. Whole group analyses.

3.7.1.1. Code acceptability judgment task. Average acceptability judgment rates split by code type for all participants are provided in Table 8 and Fig. 20. As expected, there was a main effect of condition, $F(1.22, 53.71) = 156.36$, $p < .001$. There was also a main effect of code type, $F(1, 44) = 20.69$, $p < .001$, and a condition x code type interaction, $F(1.95, 85.58) = 4.16$, $p = .020$. Holm-Bonferroni adjusted post-hoc tests revealed that across all participants, variables were judged with higher acceptability rates than keywords for the well-formed condition, $t(44) = 4.46$, $p < .001$, and structurally invalid condition, $t(44) = 3.73$, $p = .002$, but not the doubly anomalous condition, $t(44) = 1.18$, $p = .491$.

When comparing conditions within variables and within keywords, participants always judged well-formed code with higher acceptability rates than structurally invalid code (both ps $< .001$) and doubly anomalous code (both ps $< .001$), while they always judged structurally invalid code with higher acceptability rates than doubly anomalous code (variables: $p < .001$; keywords; $p = .026$).

Table 8.
Average acceptability judgments for all participants ($N = 45$), split by code type.

		Mean	SD	Min	Max
Well-formed	*Variables*	82.00	22.06	0.00	100.00
	Keywords	71.27	24.26	17.00	100.00
Structurally invalid	*Variables*	31.44	28.81	5.00	95.00
	Keywords	19.56	21.05	5.00	75.00
Doubly anomalous	*Variables*	17.34	16.63	0.00	65.00
	Keywords	14.44	16.52	5.00	85.00

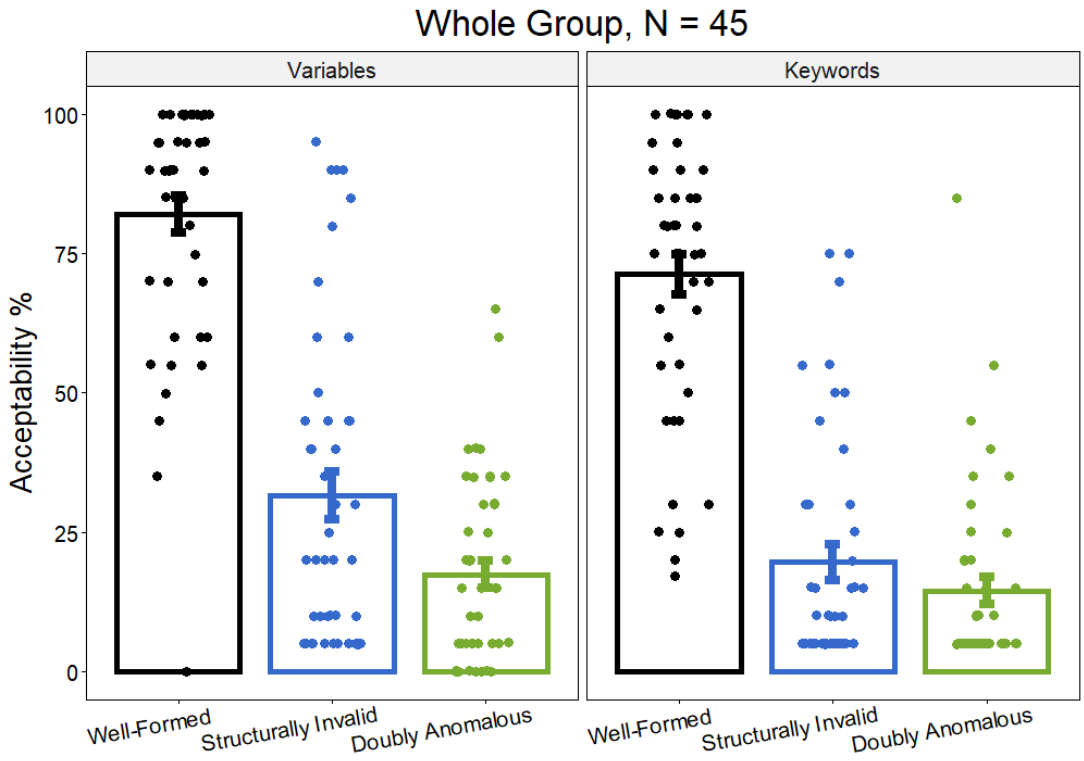

Fig. 20. Variable vs keyword acceptability judgments for all participants. Error bars represent one standard error of the mean.

3.7.1.2. Grand mean ERPs. ERPs comparing variables and keywords are plotted separately for all participants (Fig. 21).

3.7.1.2.1. N400 (300-500 ms) time window. Statistical analyses revealed a main effect of condition, $F(1.97, 86.66) = 5.15$, $p = .008$, as well as code type, $F(1, 44) = 9.23$, $p = .004$. However, there was no interaction between condition and code type, $F(1.96, 86.07) = 2.04$, $p = .138$. Holm-Bonferroni adjusted post-hoc tests revealed that relative to structurally invalid code, doubly anomalous code elicited a stronger negativity for variables, $t(44) = -3.29$, $p = .024$, but not for keywords, $t(44) = -1.40$, $p = 1.000$. No other comparisons were significant in the 300-500 ms time window except for the difference in amplitude between the structurally invalid variables and the well-formed variables. Visual inspection suggests that this is driven by a strong positivity pulling the waveform for structurally invalid variables below the well-formed variables.

3.7.1.2.2. P600 (500-800 ms) time window. Statistical analyses revealed a main effect of condition, $F(1.96, 86.17) = 24.39$, $p < .001$, as well as code type, $F(1, 44) = 6.40$, $p = .015$. However, there was no interaction between condition and code type, $F(1.99, 87.54) = 1.91$, $p = .155$. Holm-Bonferroni adjusted post-hoc tests revealed that relative to well-formed code, structurally invalid code and doubly anomalous code elicited stronger positivities for variables ($t(44) = 4.23$, $p = .001$ and $t(44) = 3.59$, $p = .007$ respectively) and for keywords ($t(44) = 5.57$, $p = < .001$ and $t(44) = 4.70$, $p < .001$ respectively). No other comparisons were significant in the 500-800 ms time window.

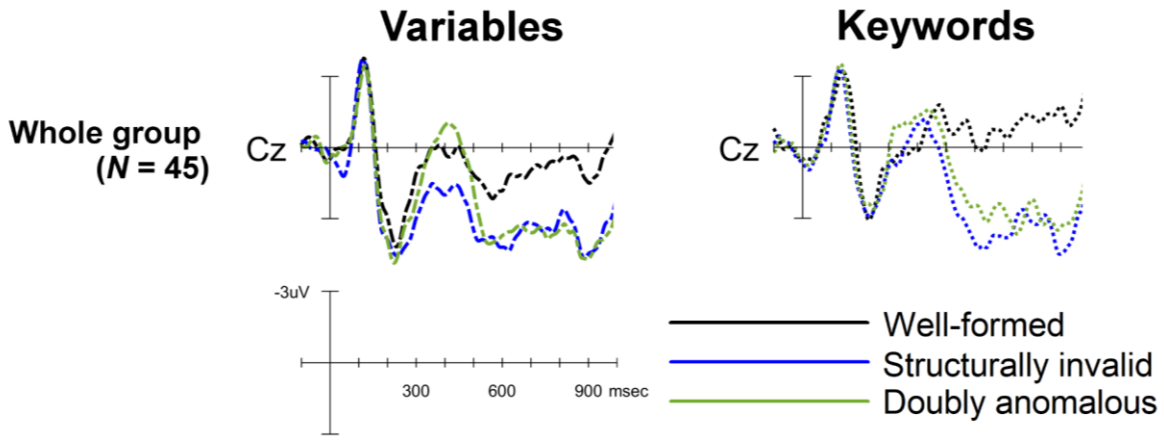

Fig. 21. Grand mean ERP waveforms for all participants ($N = 45$) comparing well-formed code, structurally invalid code, and doubly anomalous code at the Cz electrode, with variable and keyword trials plotted separately. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

3.7.2. Python Experts versus Python Novices

3.7.2.1. Acceptability judgment task. Average acceptability judgment rates split by code type for Python Experts and Python Novices are provided in Table 9 and Fig. 22. As expected, there was an interaction between condition and expertise, $F(1.30, 55.92) = 20.42$, $p < .001$. There was also a three-way interaction between condition, code type, and expertise, $F(1.99, 85.64) = 5.49$, $p = .006$, but no interaction between code type and expertise only, $F(1, 43) = 1.83$, $p = .183$. Compared to Novices, Experts judged structurally invalid variables to be less acceptable, $t(43) = -5.28$, $p < .001$, whereas the comparison of structurally invalid keywords did not survive Holm-Bonferroni adjustment, $t(43) = -3.24$, $p = .054$. Experts also judged doubly anomalous code to be less acceptable compared to Novices for variables, $t(43) = -3.43$, $p = .034$, and for keywords, $t(43) = -3.51$, $p = .028$. Experts and Novices did not differ in their judgments of well-formed variables or keywords, $ps > .77$.

Follow-up analyses with Experts and Novices in separate ANOVA models (Bonferroni adjusted family-wise alpha = .025 each) revealed differences in acceptability rates within each

Python expertise group. In Experts, there was an expected main effect of condition, $F(1.24, 32.22) = 214.68$, $p < .001$, and a main effect of code type, $F(1, 26) = 12.06$, $p = .002$, but no interaction between condition and code type, $F(1.59, 41.47) = 2.45$, $p = .110$. Holm-Bonferroni adjusted post-hoc tests revealed that Experts judged variables with a higher acceptability rate than keywords for the well-formed condition, $t(26) = 3.85$, $p = .005$, but not for the structurally invalid or doubly anomalous conditions, ps >.370. When comparing conditions within variables and within keywords, Experts judged well-formed code with higher acceptability rates than structurally invalid code for both variables and keywords, ps <.001, but considered structurally invalid code and doubly anomalous code to be similarly acceptable for both code types, ps >.150.

In Novices, there were main effects of condition, $F(1.38, 23.43) = 39.37$, $p < .001$, and code type, $F(1, 17) = 9.73$, $p = .006$, as well as a condition by code type interaction, $F(1.52, 25.90) = 7.35$, $p = .005$. Holm-Bonferroni adjusted post-hoc tests revealed that Novices judged variables with a higher acceptability rate than keywords for the structurally invalid condition, $t(17) = 5.05$, $p < .001$, but not for the well-formed or doubly anomalous conditions, ps >.160. When comparing conditions within variables and within keywords, Novices considered well-formed code to be more acceptable than structurally invalid code for keywords, $t(17) = 5.53$, $p < .001$, but not for variables, $p = .031$. Novices considered well-formed code to be more acceptable than doubly anomalous code for both variables and keywords, ps <.001. Finally, Novices considered structurally invalid code to be more acceptable than doubly anomalous code for variables, $p < .001$, but not for keywords, $p = .378$.

Table 9.
Average acceptability judgments for Python Experts ($N = 27$) and Python Novices ($N = 18$), split by code type.

		Mean	SD	Min	Max
		Python Experts (N = 27)			
Well-formed	*Variables*	87.04	17.45	45.00	100.00
	Keywords	75.44	25.69	17.00	100.00
Structurally invalid	*Variables*	16.85	18.61	5.00	85.00
	Keywords	12.04	15.21	5.00	75.00
Doubly anomalous	*Variables*	11.12	11.87	0.00	35.00
	Keywords	8.15	9.21	5.00	45.00
		Python Novices (N = 18)			
Well-formed	*Variables*	74.44	26.34	0.00	100.00
	Keywords	65.00	21.07	20.00	95.00
Structurally invalid	*Variables*	53.33	27.81	5.00	95.00
	Keywords	30.83	23.84	5.00	75.00
Doubly anomalous	*Variables*	26.67	18.63	0.00	65.00
	Keywords	23.89	20.48	5.00	85.00

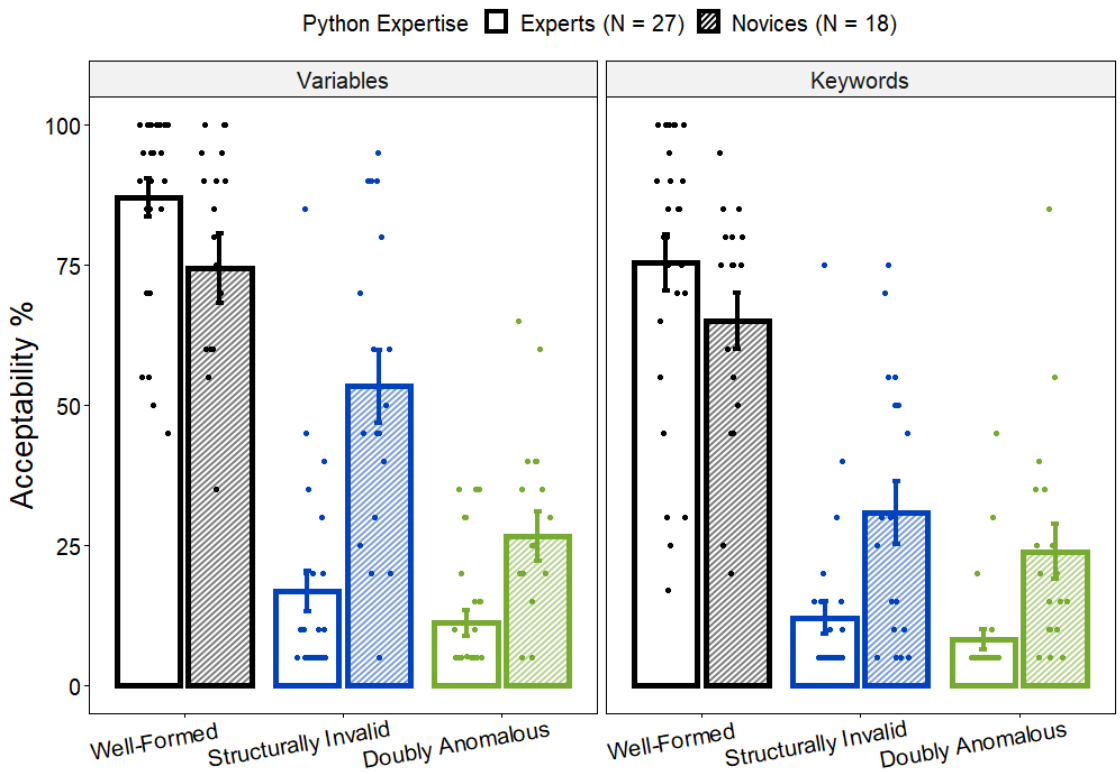

Fig. 22. Variable vs keyword acceptability judgments split by Python expertise. Error bars represent one standard error of the mean.

3.7.2.2. Grand mean ERPs. ERPs comparing variables and keywords are plotted separately for Python Experts and Python Novices (Fig. 23).

3.7.2.2.1. N400 (300-500 ms) time window. Statistical analyses revealed an expected interaction between condition and expertise, $F(1.90, 81.62) = 4.11$, $p = .022$. However, there was no interaction between code type and expertise, $F(1, 43) = 1.44$, $p = .236$, or three-way interaction between condition, code type, and expertise, $F(1.38, 6.64) = 0.21$, $p = .808$. Follow-up analyses with Experts and Novices in separate ANOVA models (Bonferroni adjusted family-wise alpha = .025 each) revealed differences in effects within each Python expertise group. For Experts, there were main effects of condition, $F(1.97, 51.17) = 7.21$, $p = .002$, and code type, $F(1, 26) = 9.65$, $p = .005$, but no interaction between condition and code type, $F(1.93, 50.30) = 2.08$, $p = .137$. Holm-Bonferroni adjusted post-hoc tests revealed that Experts exhibited a

stronger negativity to doubly anomalous variables relative to structurally invalid variables, $t(26) = 4.85$, $p < .001$. No other comparisons were significant for Experts except for the difference in amplitude between the structurally invalid variables and the well-formed variables. Visual inspection suggests that this is driven by a strong positivity pulling the waveform for structurally invalid variables below the well-formed variables for Experts. On the contrary, there were no significant main effects (condition: $p = .088$ and code type: $p = .327$) or interaction between condition and code type ($p = .715$) for Novices.

3.7.2.2.2. P600 (500-800 ms) time window. Statistical analyses revealed an expected interaction between condition and expertise, $F(1.99, 85.71) = 4.41$, $p = .015$. However, there was no interaction between code type and expertise, $F(1, 43) = 0.13$, $p = .718$, or three-way interaction between condition, code type, and expertise, $F(1.99, 85.56) = 0.07$, $p = .934$. Follow-up analyses with Experts and Novices in separate ANOVA models (Bonferroni adjusted family-wise alpha = .025 each) revealed differences in effects within each Python expertise group. For Experts, there was a main effect of condition, $F(1.96, 50.84) = 30.56$, $p < .001$, but no main effect of code type, $F(1, 26) = 4.12$, $p = .053$, or interaction between condition and code type, $F(1.94, 50.43) = 0.88$, $p = .417$. Holm-Bonferroni adjusted post-hoc tests revealed that Experts exhibited stronger positivities to structurally invalid code and doubly anomalous code relative to well-formed code for variables ($t(26) = 4.90$, $p < .001$ and $t(26) = 4.16$, $p = .003$ respectively) and for keywords ($t(26) = 5.37$, $p = < .001$ and $t(26) = 5.26$, $p < .001$ respectively). On the contrary, there were no significant main effects (condition: $p = .079$ and code type: $p = .160$) or interaction between condition and code type ($p = .372$) for Novices.

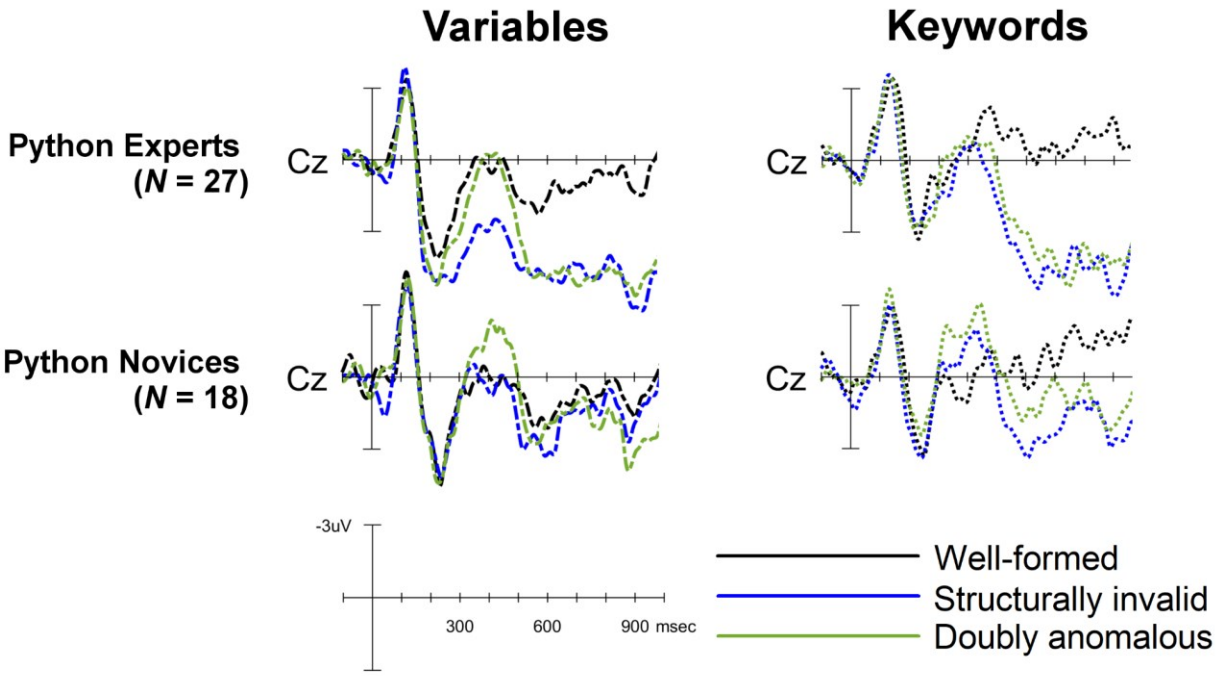

Fig. 23. Grand mean ERP waveforms for Python Experts ($N = 27$) and Python Novices ($N = 18$) comparing well-formed code, structurally invalid code, and doubly anomalous code at the Cz electrode, with variable and keyword trials plotted separately. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

3.7.3. English-High versus English-Moderate Individuals

3.7.3.1. Code acceptability judgment task. Average acceptability judgment rates split by code type for English-High individuals and English-Moderate individuals are provided in Table 10 and Fig. 24. There was no interaction between condition and English reading comprehension ability, $F(1.25, 48.72) = 2.17$, $p = .143$, code type and English reading comprehension ability, $F(1, 39) = 0.15$, $p = .696$, or three-way interaction between condition, code type, and English comprehension ability, $F(1.90, 74.14) = 1.73$, $p = .186$. Compared to English-Moderate individuals, English-High individuals judged structurally invalid keywords and doubly anomalous variables with lower acceptability, but these differences did not survive Holm-Bonferroni adjustments for multiple comparisons, $p = .700$ ($p = .036$ uncorrected) and $p = .053$

(p = .002 uncorrected) respectively. All other head-to-head comparisons between English-High individuals and English-Moderate individuals were ps >.850.

Follow-up analyses with English-High individuals and English-Moderate individuals in separate ANOVA models (Bonferroni adjusted family-wise alpha = .025 each) revealed differences in acceptability judgments within each English reading comprehension ability group. In English-High individuals, there were main effects of condition, $F(1.13, 21.46) = 75.49$, p <.001, and code type, $F(1, 19) = 9.46$, p = .006, as well as an interaction between condition and code type, $F(1.98, 37.54) = 6.73$, p = .003. Holm-Bonferroni adjusted post-hoc tests revealed that English-High individuals judged variables with higher acceptability rates than keywords for the well-formed condition, $t(19) = 4.14$, p = .004, but not for the structurally invalid condition, p = .028, or doubly anomalous condition, p = .926. When comparing conditions within code types, English-High individuals considered well-formed code to be more acceptable than structurally invalid code and doubly anomalous code for both variables and keywords, all ps <.001. On the contrary, English-High individuals considered structurally invalid code and doubly anomalous code to be similarly acceptable for variables, p = .028, and for keywords, p = .926.

In English-Moderate individuals, there were main effects of condition, $F(1.40, 28.01) = 66.59$, p <.001, and code type, $F(1, 20) = 8.13$, p = .010, but no interaction between condition and code type, $F(1.74, 34.75) = 0.78$, p = .451. Holm-Bonferroni adjusted post-hoc tests revealed that English-Moderate individuals judged variables and keywords with similar acceptability rates for all three conditions, well-formed: $t(20) = 1.99$, p = .181; structurally invalid: $t(20) = 2.20$, p = .158; doubly anomalous: $t(20) = 1.79$, p = .181. When comparing conditions within variables and within keywords, English-Moderate individuals considered well-formed code to be more acceptable than structurally invalid code and doubly anomalous code for both variables and

keywords, all *p*s <.001. On the contrary, English-Moderate individuals considered structurally invalid code and doubly anomalous code to be similarly acceptable for variables, $p = .056$, and for keywords, $p = .128$.

Table 10.
Average acceptability judgments for English-High individuals ($N = 20$) and English-Moderate individuals ($N = 21$), split by code type.

		Mean	**SD**	**Min**	**Max**
		English-High (N = 20)			
Well-formed	*Variables*	82.25	26.03	0.00	100.00
	Keywords	68.50	27.68	20.00	100.00
Structurally invalid	*Variables*	22.25	24.68	5.00	90.00
	Keywords	12.50	16.58	5.00	75.00
Doubly anomalous	*Variables*	9.76	11.52	0.00	35.00
	Keywords	10.75	17.94	5.00	85.00
		English-Moderate (N = 21)			
Well-formed	*Variables*	79.29	19.45	35.00	100.00
	Keywords	71.29	22.72	17.00	100.00
Structurally invalid	*Variables*	39.29	29.12	5.00	95.00
	Keywords	26.19	23.12	5.00	75.00
Doubly anomalous	*Variables*	24.05	15.54	0.00	60.00
	Keywords	17.86	15.46	5.00	55.00

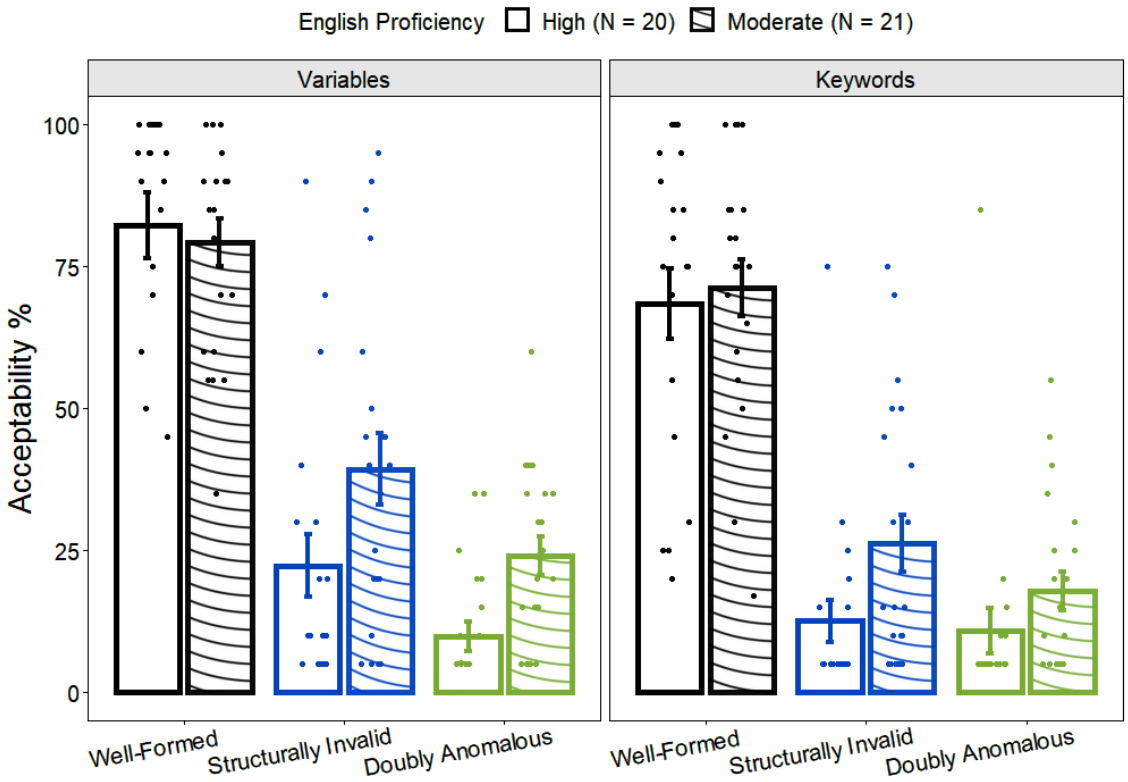

Fig. 24. Variable vs keyword acceptability judgments split by English reading comprehension ability. Error bars represent one standard error of the mean.

3.7.3.2. Grand mean ERPs. ERPs comparing variables and keywords are plotted separately for English-High individuals and English-Moderate individuals (Fig. 25).

3.7.3.2.1. N400 (300-500 ms) time window. Statistical analyses revealed no interaction between condition and English reading comprehension ability, $F(1.97, 76.89) = 2.04$, $p = .138$, code type and English reading comprehension ability, $F(1, 39) = 0.23$, $p = .637$, or three-way interaction between condition, code type, and English reading comprehension ability, $F(1.99, 77.80) = 0.17$, $p = .845$. Follow-up analyses with English-High individuals and English-Moderate individuals in separate ANOVA models (Bonferroni adjusted family-wise alpha = .025 each) revealed differences in effects within each English reading comprehension ability group. For English-High individuals, there were no significant main effects (condition: $p = .322$ and code type: $p = .118$) or interaction between condition and code type ($p = .152$). On the contrary,

English-Moderate individuals showed main effects of condition, $F(1.99, 39.74) = 4.54$, $p = .017$, and code type, $F(1, 20) = 7.32$, $p = .014$, but no interaction between condition and code type, $F(2, 39.93) = 1.02$, $p = .371$. Holm-Bonferroni adjusted post-hoc tests revealed that English-Moderate individuals exhibited a stronger negativity to doubly anomalous variables relative to structurally invalid variables, $t(20) = -3.72$, $p = .019$. No other comparisons were significant in the 300-500 ms time window for either group.

3.7.3.2.2. P600 (500-800 ms) time window. Statistical analyses revealed no interaction between condition and English reading comprehension ability, $F(1.97, 76.87) = 2.04$, $p = .107$, code type and English reading comprehension ability, $F(1, 39) = 0.23$, $p = .634$, or three-way interaction between condition, code type, and English reading comprehension ability, $F(1.97, 76.85) = 0.11$, $p = .892$. Follow-up analyses with English-High individuals and English-Moderate individuals in separate ANOVA models (Bonferroni adjusted family-wise alpha = .025 each) revealed differences in effects within each English reading comprehension ability group. For English-High individuals, there was a main effect of condition, $F(1.94, 36.91) = 15.71$, $p < .001$, but no main effect of code type, $F(1, 19) = 1.74$, $p = .203$, or interaction between condition and code type, $F(1.88, 35.71) = 0.51 = .594$. Holm-Bonferroni adjusted post-hoc tests revealed that English-High individuals exhibited stronger positivities to structurally invalid keywords and doubly anomalous keywords relative to well-formed keywords ($t(19) = 4.02$, $p = .009$ and $t(19) = 4.72$, $p = .002$ respectively). English-High individuals also exhibited stronger positivities to doubly anomalous variables relative to well-formed variables, $t(19) = 3.65$, $p = .021$, while the difference between structurally invalid variables and well-formed variables did not survive corrections for multiple comparisons, $t(19) = 2.88$, $p = .077$ ($p = .010$ uncorrected). Meanwhile, there was also a significant main effect of condition for English-Moderate

individuals, $F(1.93, 38.59) = 9.89$, $p < .001$, but no main effect of code type, $F(1, 20) = 5.04$, $p = .036$, or interaction between condition and code type, $F(1.79, 35.80) = 1.40$, $p = .258$. English-Moderate individuals displayed stronger positivities to structurally invalid variables relative to well-formed variables, but this difference did not survive Holm-Bonferroni adjustment, $t(20) = 3.49$, $p = .032$ ($p = .002$ uncorrected). Similarly, English-Moderate individuals exhibited stronger positivities to structurally invalid keywords and doubly anomalous keywords relative to well-formed keywords that did not survive Holm-Bonferroni adjustment, $t(20) = 3.25$, $p = .052$ ($p = .004$ uncorrected) and $t(20) = 2.68$, $p = .157$ ($p = .014$ uncorrected) respectively. No other comparisons were significant in the 500-800 ms time window for either group.

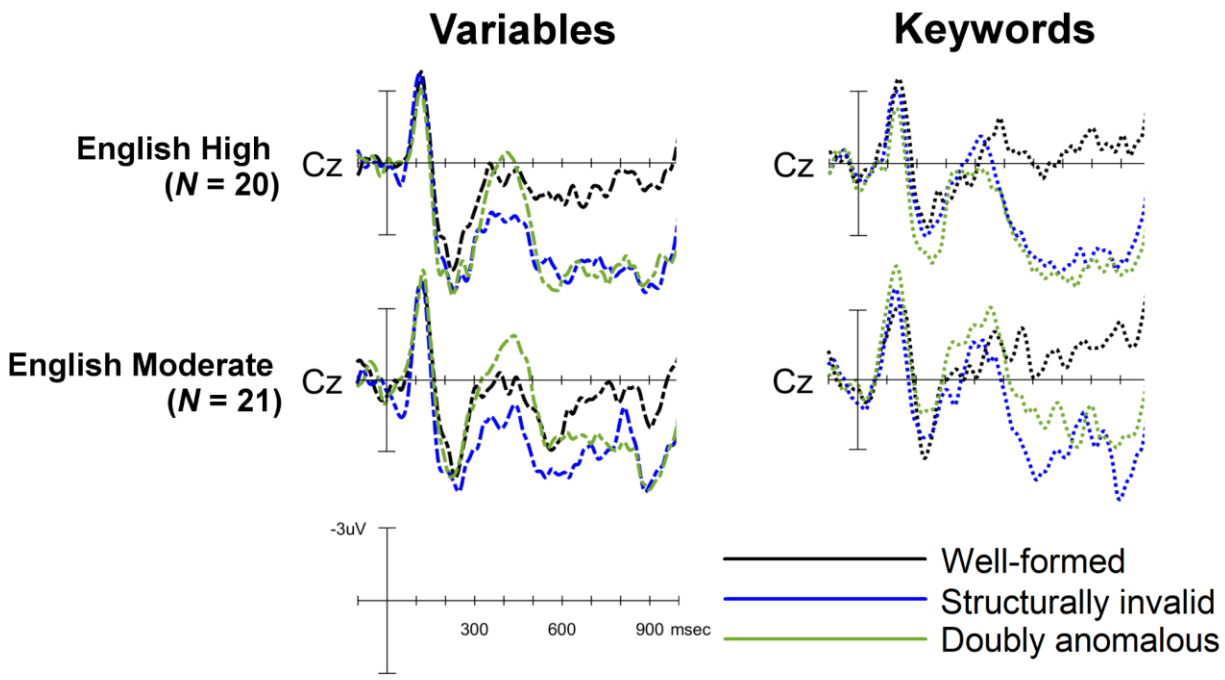

Fig. 25. Grand mean ERP waveforms for English-High individuals ($N = 20$) and English-Moderate ($N = 21$) comparing well-formed code, structurally invalid code, and doubly anomalous code at the Cz electrode, with variable and keyword trials plotted separately. Onset of the target unit of code in a given trial is indicated by the vertical bar. Calibration bar shows 3 μV of activity. Negative voltage is plotted up.

Discussion

The goal of this dissertation was to investigate the creation of mental models that underlie computer programming by observing programmers' ERPs to well-formed and anomalous Python code. The reported results are the first evidence that meaning and form jointly influence mental model creation, although the timing, behavioral sensitivity, and neural responses to violations vary with coding expertise. In general, the findings are consistent with the prediction that, like other symbolic cognitive systems, programming expertise is centered around the construction, retrieval, and application of schemas. Below, this evidence is discussed in further detail, along with its implications for future research.

4.1. Meaning and Form Jointly Influence Mental Model Creation During Code Comprehension

The data presented in this dissertation provide converging evidence that code comprehension is driven by information about how meaning is gleaned from the goal of the code as it is constructed. In the following sections, I argue that three main findings contribute to the claim that code comprehension, similar to other symbolic systems such as natural language and mathematics, relies on the integration of meaning and form during stimulus processing to construct a mental model of the program.

4.1.1. Programmers show distinct neural signatures to violations of meaning and form. At the group level, participants exhibited distinct ERPs in response to violations of meaning and form during code comprehension. Relative to their well-formed counterparts, violations of meaning (semantically implausible code) elicited N400 effects and violations of form (structurally invalid code) elicited P600 effects, with stronger positivities recorded over posterior electrode locations. These responses mirror the distinct ERPs that humans exhibit to

anomalies in natural language; specifically, N400 effects to errors in semantics and P600 effects to errors in syntax that are more prominent toward the posterior (Osterhout & Nicol, 1999). Although the present study only analyzed a subset of the midline electrodes, the topography of the effects suggest that code comprehension may be generated by similar mechanisms to classic natural language-based mechanisms. Similar N400/P600 patterns have been observed in mathematics (Galfano et al., 2004; Niedeggen et al., 1999; Núñez-Peña & Honrubia-Serrano, 2004) and music (Calma-Roddin & Drury, 2020; Besson & Macar, 1987; Janata, 1995; Patel et al., 1998), providing support for the hypothesis that programming languages are processed similarly to other symbolic systems (Fedorenko et al., 2019). Source localization would be a logical next step for examining the extent to which code comprehension shares cognitive mechanisms with natural language specifically, compared to mathematics, music, or other symbolic systems.

The present findings suggest that programmers engage in real-time processing and manipulation of incoming information to comprehend code. Their ERPs are reflective of their expectations being violated, which occurs if programmers are attempting to construct a mental model that integrates meaning and form to predict the outcome of a snippet of code (Luck, 2005). Of note is the fact that programmers' responses to code anomalies have different time courses. Retrieval of semantic information occurs first (Kutas & Federmeier, 2014; Lau et al., 2008), followed by its integration into the overall structure with respect to syntax rules of the programming language (Aurnhammer et al., 2021, 2023; Swaab et al., 2012). These neural signatures were observed in response to the unit of code containing the violation, which occurs in the middle of a line of code. Therefore, programmers do not wait until the entire program has been presented to process the individual units; rather, they build an internal representation of the

goal of the code as they obtain more information, adjusting as needed to mismatches in meaning or form. The more effort required to retrieve meaning or resolve unexpected structural configuration as a result of the anomaly present in a unit of code, the stronger an individual's N400 and P600 effects, respectively.

4.1.2. Programmers judge violations of meaning and form with different acceptability rates. Violations of meaning and form also influenced participants' code acceptability judgments. Although participants considered code containing any type of violation to be less acceptable than well-formed code, they also considered code containing form violations only (structurally implausible code) to be less acceptable than code containing meaning violations only (semantically implausible code). Furthermore, code containing both a meaning and form violation (doubly anomalous code) was considered the least acceptable.

Together with participants' ERP responses, these data suggest that programmers process meaning and form as unique sources of information during code comprehension. While semantic implausibility alone does not prohibit code from being compiled (see section 4.1.3 below), structural invalidity always results in uncompilable code. Importantly, both types of violations impact programmers' acceptability judgments in differing degrees depending on whether the violations are present individually or in combination. Therefore, programmers are aware of the extent to which meaning and form violations impact the overall soundness of a program, an indication that their mental model is constructed by integrating information about meaning and form into a broader understanding of the code instead of treating them separately. Otherwise, semantically implausible code and structurally implausible code would be judged with the same acceptability rates, and there would not be distinct neural signatures to the two types of violations.

4.1.3. Manipulations of different code types provide insight into the mental models that programmers construct during code comprehension. In the present study, semantically implausible trials were legally formed code (i.e., the code would compile because it adhered to the rules of syntax) in which the manipulated unit of code was a variable name that acted as an iterator through a referenced global variable, or iterable (e.g., list, dictionary). Variables are similar to nouns that set the semantic context for a segment of code. A global variable-iterator variable pair that has low semantic association is akin to word pairs that have low semantic association (e.g., *doctor-pear* versus *doctor-nurse*). Previous research has found that words can prime faster and more accurate recognition of subsequent words when they are semantically related rather than semantically unrelated (Meyer & Schvaneveldt, 1971; Neely, 1991); furthermore, unrelated word pairs elicit larger N400 effects than related word pairs (Bentin et al., 1985; Holcomb, 1988; Holcomb & Neville, 1990; Kutas & Hillyard, 1989). As such, the finding that semantically unrelated variable names induced lower acceptability rates and increased N400 effects relative to semantically related variable names suggests that programmers use the semantics evoked by variables as linguistic cues. These cues guide programmers to retrieve a schema that fits the intended theme of the code, which supports their mental model construction for the overall goal. In doing so, they attach purpose to the actions carried out by the code, such that the code becomes a meaningful entity rather than a random operation. For example, when a programmer sees the global variable *pets = [“dog”, “cat”, “hamster”]* followed by the code *for every animal in the list pets: print(animal)*, they may infer that there is an animal shelter that has pets up for adoption, or that someone owns multiple pets, and the code provides a list of which animals are available. Alternatively, if the *pets* global variable was followed by the code *for every fruit in the list pets: print(fruit)*, the programmer’s schema of “an animal shelter contains

pets for adoption" or "a pet owner owns multiple animals" would be violated because pets and fruit are not typically associated with each other. This interrupts their mental model and overall understanding of the goal of the code. Therefore, not only do programmers rely on schemas to build the semantic content for the mental model of a segment of code, the degree to which English semantics is central to the identity of a unit of code also affects parsing. Previous research has found that linguistic cues similarly affect the situation models that comprehenders build during natural language processing, in which sentences that refer to the same agents, time, actions, entities, goal, etc. make them easier to integrate into a larger narrative (e.g., Zwaan, 2016; Zwaan & Radvansky, 1998).

One caveat to this interpretation is that these results may be a byproduct of semantic priming alone rather than meaning making during code comprehension per se, especially since participants were not asked to report the output of code. To tease apart these effects, an additional manipulation could be added in which the variable name was a semantically related, but functionally unsuitable word. For example, consider a trial in which the *pets* global variable was followed by the code *for every bird in the list pets: print(bird)*. If the present results were solely due to semantic priming, the *bird* variable name would elicit similar effects to the ones reported herein. However, if the present results were due to meaning making, the *bird* variable name could elicit even lower acceptability rates and more prominent N400 effects because *bird* does not accurately describe every item in the list *pets*.

Another potential confound with the reported findings is that the semantically implausible condition only manipulated variables, whereas the structurally invalid condition manipulated variables and keywords. This was incorporated into the design in order to provide variability in stimulus, but it also led to an imbalance in code type trials when the conditions

were crossed. If variables can be thought of as nouns, then keywords can be thought of as verbs that serve very precise roles. Unlike variables, keywords are reserved words that carry out specific actions, and they cannot be used as variable names, function names, or other identifiers. Importantly, they are built-in entities of a programming language and cannot be replaced with any other English word. Therefore, manipulations of keywords are inherently structurally invalid and cannot be semantically implausible only, even if a suitable English synonym is used. Due to this rigidity, when the role of keywords is violated or not fulfilled, it could be interpreted as an unexpected continuation due to a violation in structure. As a result, it is reasonable to ask whether the reported findings for the semantically implausible condition is bolstered by the presence of keywords in the structurally invalid and doubly anomalous conditions.

To address this possibility, trials were further broken up into variables and keywords for the structurally invalid and doubly anomalous conditions and compared to one another. In addition to reducing the potentially confounding effect of imbalanced trial types, this set of analyses was carried out to explore the degree to which the rigidity of English semantics as they relate to the type of code being manipulated influenced individuals' code acceptability judgments and ERPs. Regardless of the code type, participants judged well-formed code with higher acceptability rates than structurally invalid code, which was judged with higher acceptability rates than doubly anomalous code. However, variables were judged with higher acceptability rates than keywords for both the well-formed and structurally invalid conditions, but not the doubly anomalous conditions. These findings seem to reflect an interaction between meaning and form in Python code. Specifically, while the function of a variable in a line of code in Python may be reflected by the name chosen for that variable, that name is ultimately irrelevant to the syntax of the code. As such, semantics and structure can be independently manipulated for

variable names, and their soundness is not tied to each other. This introduces considerable flexibility around how variable names are chosen, which makes semantics more central to the identity of variables. In contrast, there are a finite number of legal keywords in Python, with 35 total as of Python version 3.8 (Hansen, n.d.). All of these keywords provide semantic clues about what their function is, but this relationship is inflexible; replacing keywords with synonyms will cause the code not to run (e.g., replacing *False* with *Untrue*). Therefore, structure is more central to the identity of keywords.

If variables are defined by their semantic soundness, and keywords are defined by their structural soundness, then semantics may serve as more of a facilitating cue when judging the acceptability of well-formed variables compared to keywords. For example, when a participant sees the *pets* global variable followed by *for every animal in the list pets: print(animal)*, the acceptability of the iterator variable *animal* is facilitated by its semantic association with *pets*, but the keyword *in* does not benefit from a similar relationship. Therefore, participants need to rely on their own knowledge of Python to determine the acceptability of well-formed keywords, whereas they may be aided by the semantics of the global variable when determining the acceptability of well-formed variables. Along the same lines, manipulating the structural validity of variables introduces a punctuation error attached to a semantically plausible iterator, whereas manipulating the structural validity of keywords replaces the correct keyword with an English synonym. A non-existent keyword, regardless of how closely it is replaced, may be more easily recognized as unacceptable than quotations that do not belong. Furthermore, the fact that structurally invalid variables still maintain strong semantic associations with the preceding global variables may accidentally lure participants into believing they are acceptable more often than not. This would explain why acceptability rates were similarly low for doubly anomalous

variables and keywords. By removing the semantic association between the global variable and the iterator, on top of introducing a punctuation error, the semantic soundness and structural soundness are in sync. This consistency helps participants rate doubly anomalous variables as unacceptable. Meanwhile, a non-existent keyword that is replaced by an English word with a completely irrelevant meaning also compounds the unacceptability of the code. As such, code type influences the processing of meaning and form during code comprehension.

ERPs provide further support for this idea. Relative to well-formed code, participants exhibited P600 effects of similar magnitude to structurally invalid and doubly anomalous code for both variables and keywords. This indicates that participants were neurally sensitive to structural violations for both code types (punctuation errors for variables and non-existent keywords). However, relative to structurally invalid code, participants exhibited stronger N400 effects to doubly anomalous code for variables, whereas there was no observed N400 effect to keywords. Therefore, even though participants' judged doubly anomalous variables and keywords to be similarly egregious, their neural sensitivity to their semantic implausibility is characterized by different brain responses. Semantic expectation is *specifically* violated when a global variable-iterator pair is not semantically related, leading participants to register this error as an anomaly in meaning. On the other hand, breaking the structure of keywords necessitates breaking their semantics; as such, participants register all errors related to keywords as an anomaly in structure, regardless of their semantic soundness. In other words, in situations where English semantics was not as fixed, participants were more likely to process meaning violations as form violations. This is similar to the "semantic attraction" effect in natural language when the thematic role of a verb as implied by the provided context is inconsistent with the role that it usually inhabits, eliciting a P600 effect rather than the traditional N400 effect associated with

semantic anomalies (Kim & Osterhout, 2005; Kuperberg, 2007). Overall, these results indicate that, similar to natural language, the semantics-syntax dynamic is fluid in code comprehension. Processing of meaning and form often depends on the way they relate to each other in the context of the specific code structure, as well as how central English semantics is to its soundness.

4.2. Expert and Novice Programmers Differ in Their Sensitivity to Meaning and Form

The results from this study also provided novel insights about the relationship between programming expertise and mental model creation during code comprehension. Specifically, expert programmers are more sensitive to structural violations, similar to natural language processing in highly proficient users. Additionally, programming expertise determines an individual's sensitivity to semantic information during code comprehension depending on the degree to which that information is central to the identity of the code.

4.2.1. Programming expertise is indicated by structural sensitivity during code comprehension. Expert and novice programmers differed in both their behavioral acceptability judgments and in their ERPs, particularly in their sensitivity to structural violations. Compared to Novices, Experts judged lines of code with structural violations (i.e., structurally invalid and doubly anomalous code) to be less acceptable. Furthermore, Experts considered structurally invalid code to be as egregious as doubly anomalous code, which were both considered less acceptable than well-formed code or semantically implausible code. This is different from Novices, who considered structurally invalid code and semantically implausible code to be similarly acceptable, which were both considered more acceptable than doubly anomalous code. These results suggest that Experts were more sensitive to structural cues during code comprehension and that they viewed structural validity as the more important factor in

determining the acceptability of code. Meanwhile, semantic plausibility of code did not influence their behavioral judgments, suggesting that Experts were aware that the semantically implausible code they encountered were structurally legal according to the rules of the language syntax. Altogether, Experts appear to treat meaning and form as distinct pieces of information. On the other hand, Novices do not distinguish between violations of meaning and form on paper, suggesting that they place an equal weight on both violation types regardless of their impact on the output. Furthermore, the fact that Novices judged doubly anomalous code at roughly half of the acceptability rate of either structural or semantic violations alone indicates that they treat violations as additive. The more violations present in a line of code, the more cues they can draw upon to comprehend code.

The differing neural responses of Experts and Novices to violations of meaning and form also indicate that sensitivity to structure is the hallmark of programming expertise. Experts exhibited stronger P600 effects to structurally invalid code compared to Novices, while they exhibited N400 effects of similar magnitudes to semantically implausible code. Taken together, these results suggest that programmers of all skill levels recognize when a unit of code is thematically incompatible with the rest of the code when building their mental models, but programmers with more expertise do not consider meaning violations to be detrimental to the overall goal of the code. Instead, attention to code structure, which requires rapid integration of structural information, is what separates the mental models of Experts and Novices. Novices display some sensitivity to structural violations on paper and in their neural signatures, but these responses are weaker compared to Experts. Therefore, programming expertise may be indexed by the magnitude of one's P600 to form violations rather than meaning violations during code comprehension. This finding is similar to native speakers' tendency to display P600 effects to

syntax errors while L2 learners initially display N400 effects or weak P600 effects during early stages of acquisition (McLaughlin et al., 2010). Overall, these results provide evidence that expert and novice programmers differ in the extent to which their schemas of semantic world knowledge and knowledge of conceptual syntax rules jointly influence the organization of their mental models (Mayer, 1981; Robins et al., 2003).

To further explore the differences in structural sensitivity between expert and novice programmers, two *d'* metrics were calculated for each participant based on structural validity as the desired signal of interest:

1. An individual's sensitivity to structural validity when the code is semantically plausible. Hit rate was computed as the proportion of "Yes" responses to well-formed code. False alarm rate was computed as the proportion of "Yes" responses to structurally invalid code.
2. An individual's sensitivity to structural validity when the code is semantically implausible. Hit rate was computed as the proportion of "Yes" responses to semantically implausible code. False alarm rate was computed as the proportion of "Yes" responses to doubly anomalous code.

Experts had higher *d'* scores than Novices for both measures, but the two measures did not differ within either expertise group, with statistical analyses supporting this observation (Fig. 26). This suggests that programming expertise drives sensitivity to structure even when semantic plausibility is held constant, supported by the fact that there is no major difference between the distributions of the two *d'* measures when collapsing across expertise (Fig. 27).

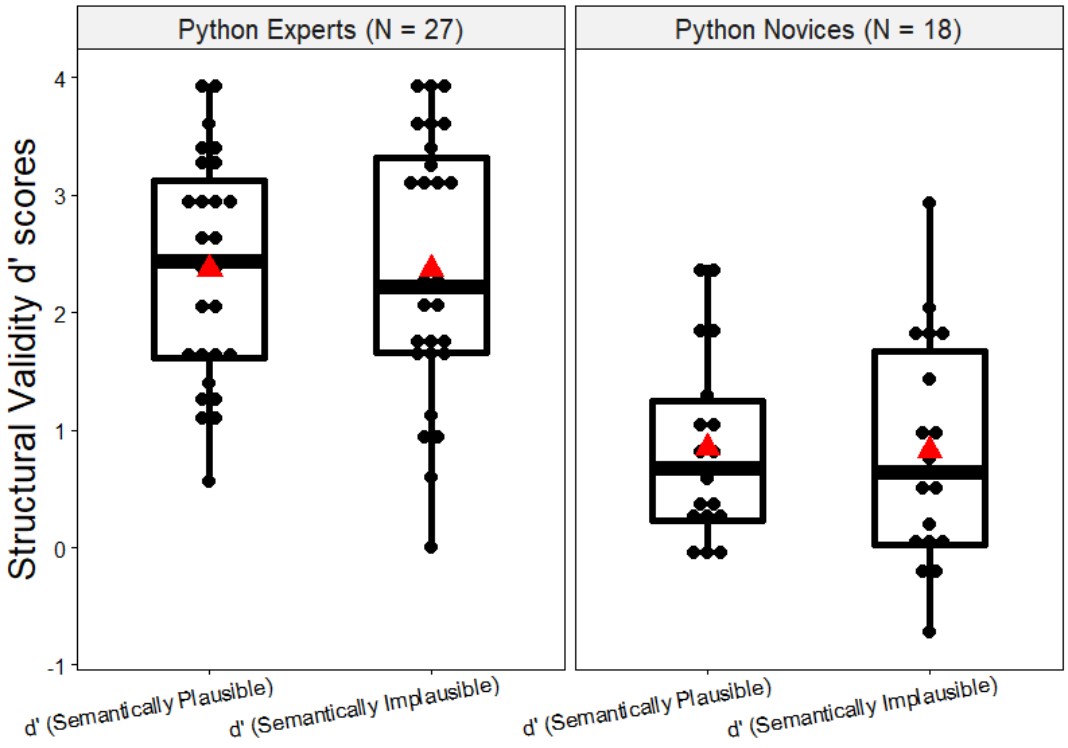

Fig. 26. *d'* scores split by Python proficiency. Triangle denotes the mean.

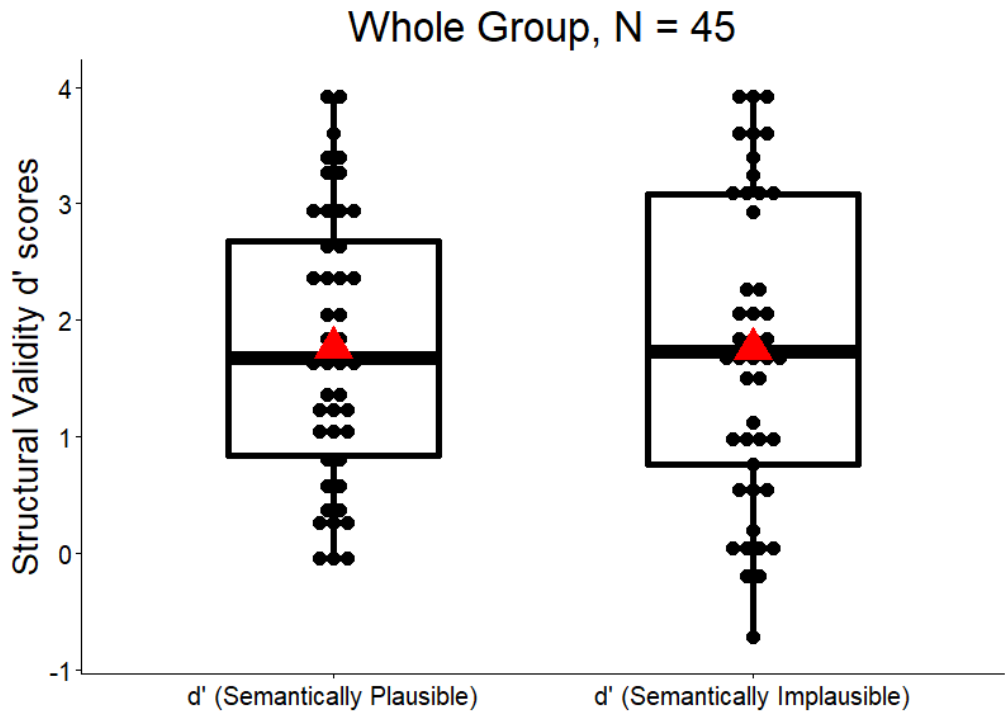

Fig. 27. *d'* scores collapsed across all participants. Triangle denotes the mean.

4.2.2. Expert and novice programmers display different sensitivities to meaning and form violations depending on the code type. Programming expertise modulated the effects of code type (variables versus keywords) described in section 4.1.3. Specifically, Experts considered anomalous code to be equally unacceptable regardless of condition or code type, whereas Novices judged variables with higher acceptability rates than keywords for structurally invalid code, but not doubly anomalous code. Furthermore, Novices considered structurally invalid code to be more acceptable than doubly anomalous code for variables but not keywords. These results suggest that Experts considered structural validity to be the key factor in determining the soundness of code, with the fluidity of the semantic constraint playing little to no part in their judgments. On the contrary, Novices relied more on top-down schemas related to the semantic content to determine the soundness of code. These patterns were supported by ERPs, in which Experts exhibited stronger P600 effects to structurally invalid and doubly anomalous code relative to well-formed code for both code types, while Novices' P600 effects were not strong enough to achieve significance. Overall, Experts judged the acceptability of code based on structure alone, and they were also more sensitive to structural violations compared to Novices, suggesting that Experts' mental models are driven by a focus on form, and Novices' mental models are driven by a focus on meaning.

However, this distinction does not imply that Experts do not register semantic anomalies at all; in fact, Experts exhibited stronger N400 effects to doubly anomalous code relative to structurally invalid code for variables, in which the semantic constraints were more fluid, but not for keywords, in which the semantic constraints were fixed. On the contrary, Novices' N400 effects to doubly anomalous code relative to structurally invalid code were not strong enough to achieve significance. Therefore, not only do Experts process meaning and form with greater

sensitivity compared to Novices, the amplitude of their responses are reflective of the fluidity of English semantics in a given coding context. This may suggest that semantics play a supporting role in the construction of mental models in more proficient programmers, for which semantics (and related schemas) guide, but do not direct, code comprehension. On the contrary, less proficient programmers rely more on semantics during code comprehension, but their neural responses are not strong enough to reflect this trait. This could be due to larger variability in the ERP responses within the Novices, in which some individuals primarily exhibit N400 effects and others primarily exhibit P600 effects. If so, this would mirror the observation that learners of a second natural language initially display N400 effects to both semantics and syntax violations before gradually developing distinct neural responses with expertise, and that there are individual differences in the rate at which learners transition to native-like responses (McLaughlin et al., 2010). A similar phenomenon could be happening in this study, in which some Novices are more advanced than others in developing their programming expertise, leading to varied neural responses to meaning and form violations when separated by code type.

Altogether, code acceptability judgments, *d'* measures, and ERPs indicate that expert and novice programmers are differently affected by semantic plausibility and structural validity during the integration of meaning and form to comprehend code. Experts are less influenced by their schemas related to semantics when evaluating code, and their deep syntax knowledge is strong enough to override cases in which inconsistent meaning schemas may interfere with their mental model construction. On the contrary, Novices have more shallow understandings of syntax and rely on their meaning schemas to build their mental models during code comprehension. However, there may be larger amounts of variability within Novices that contribute to their weaker ERP responses when trials are separated by code type; more work will

have to be done to determine if Novices transition to expert-like responses with increased experience (see Future Directions, section 4.7).

4.3. English Proficiency Weakly Resembles Python Expertise

Due to previous research indicating that natural language can facilitate the processing of semantic information (Anderson, 1977; Barnitz, 1986; Carrell, 1984), the present study also investigated the extent to which English proficiency, measured as English reading comprehension ability, impacts programmers' processing of meaning and form violations during code comprehension. Findings suggest that English proficiency mediates programmers' code acceptability judgments on paper, but not their neural signatures to meaning and form violations. Differences in neural signatures arise when trials are separated by code type. Limitations of these results are discussed.

4.3.1. English proficiency mediates programmers' code acceptability judgments but not their overall neural signatures to meaning and form violations. English proficiency influenced programmers' code acceptability judgments in a similar fashion as programming expertise. Although English-High individuals considered semantically implausible code to be less acceptable than well-formed code, which is different from expert programmers, they also judged semantically implausible code to be more acceptable than structurally invalid code, which was judged to be more acceptable than doubly anomalous code. On the contrary, English-Moderate individuals' code acceptability judgments displayed the same pattern as novice programmers; well-formed code was considered the most acceptable, followed by similar acceptability rates for semantically implausible code and structurally code, with doubly anomalous code judged as the least acceptable. These patterns would imply that individuals with lower English proficiency are more affected by semantics during code comprehension, but this

did not translate to their ERPs. Visual inspection suggests that individuals with lower English proficiency display stronger sensitivity to meaning such that their sensitivity to form is weaker, especially when violations of both types are different, but these effects did not reach significance. This would indicate that English experience does not play a significant role in how programmers process code anomalies in their brain, only how they classify them on paper. As such, programming expertise appears to be more influential in distinguishing programmers' construction of mental models than English experience. However, it is possible that English experience influences code comprehension under very specific circumstances, which is discussed below in section 4.3.2.

4.3.2. English proficiency influences neural sensitivity to meaning and form violations depending on the code type. English experience modulated programmers' neural responses to the effects of code type (variables versus keywords) described in section 4.1.3, while weakly affecting their code acceptability judgments. Specifically, differences in acceptability judgments between the English-High and English-Moderate groups, as well as comparisons within each English proficiency group that did not involve the well-formed condition, were not strong enough to survive adjustments for multiple comparisons. More prominent differences arose when comparing ERPs. English-High individuals exhibited stronger P600 effects to structurally invalid and doubly anomalous code compared to well-formed code for both code types, whereas English-Moderate individuals exhibited stronger N400 effects to doubly anomalous variables relative to structurally anomalous variables. Despite visual inspection suggesting that English-High individuals also exhibited N400 effects to doubly anomalous variables relative to structurally anomalous variables, and that English-Moderate individuals also exhibited stronger P600 effects to structurally invalid and doubly anomalous

code for keywords, these effects were not strong enough to reach significance. Altogether, individuals with higher English proficiency displayed stronger sensitivity to form violations, with the fluidity of the semantic constraint playing little to no part in their judgments. On the contrary, individuals with lower English proficiency displayed stronger sensitivity to meaning violations, but only for situations in which English semantics was more fluid.

Together, these results imply that English experience guides code comprehension under specific circumstances. It is possible that higher English proficiency enables participants to process the semantics faster or with greater efficiency, allowing them to focus on the structure of the code, which lessens the influence that world knowledge schemas interfere with the integration of meaning and form. Meanwhile, individuals with moderate or lower English proficiency employ additional effort to retrieve meaning from semantic memory, which causes them to be more impacted by mismatched English semantics when the anomaly is related to the overall thematic context of the code. Therefore, English experience can still interfere with a programmer's ability to construct a mental model to represent the ultimate goal of a chunk of code when they have poorer English proficiency. However, there are several factors that limit the interpretation of these results, which are discussed in section 4.3.3. below.

4.3.3. Potential contributing factors to English proficiency results. Despite visual inspection suggesting more prominent differences in neural responses to meaning and form violations between English-High and English-Moderate individuals, many of the effects did not reach significance. This could be due to the ceiling effect in the method that was selected to index participants' English reading comprehension ability; measurements that capture other aspects of English proficiency may provide more variability. There was also a large overlap between English-High individuals and Python Experts. Although there was no statistical

difference in Python knowledge test accuracy between the two English proficiency groups, the imbalance was large enough such that the effects of programming expertise potentially overshadowed any unique effects of English proficiency. To tease apart the effects of programming expertise and English proficiency, future studies should consider how these traits may interact within an individual rather than how they separately affect programmers' code comprehension.

An alternative explanation of the results is that the pacing of stimulus presentation influenced the degree to which programmers could leverage information about English semantics to construct their mental models. Even though the presentation rate was set to mirror slower rates utilized in ERP studies of second language acquisition (Foucart & Frenck-Mestre, 2011, 2012; Tanner et al., 2013, 2014), code may be, by default, less prone to forward prediction than natural language. Due to the unfamiliar presentation format for code (unit-by-unit rather than the entire program at once), an individual may feel forced to rely on either their programming knowledge or their English knowledge to mitigate the working memory demands to parse the code structure. This "forced selection" may be more prominent in novice programmers or individuals with lower English proficiency who have difficulty integrating meaning *and* form into a coherent mental model, leading to the large variability in code acceptability judgments and neural responses.

Research in the natural language literature on speed reading, defined as "reading at an increased speed without any loss of comprehension," indicates that there is a tradeoff between speed and accuracy (Rayner et al., 2016). If the goal of a reader is to read quickly with moderate comprehension, then speed reading is a viable method; otherwise, high comprehension of text under speed reading can only be achieved by becoming a more skilled language user. The "good-enough representation" account of language comprehension also dictates that people rarely

engage in complete processing of language, especially in cases where the syntax is complex or unfamiliar (Ferreira & Bailey, 2002). Instead, people rely on schemas cued by individual words to glean a rough meaning of text, resulting in erroneous interpretations (Karimi & Ferreira, 2016). Studies comparing different presentation rates of text support these findings, showing that faster presentation rates bias individuals towards utilizing lower-level lexical features and schemas, while slower presentation rates invite higher-level comprehension strategies (e.g., Camblin et al., 2007; Ledoux et al., 2007; Swaab et al., 2004). In light of these findings, self-paced reading may be a better method to study code comprehension, because it encourages readers to engage in processes that they typically rely on during natural language comprehension (Ditman et al., 2007).

4.4. Individual Differences in Semantic Sensitivity

Overall, neither programming expertise nor English proficiency significantly influenced neural responses to meaning violations unless trials were split by code type. This introduces the question of whether semantic sensitivity is relevant to understanding the construction of mental models by programmers during code comprehension. However, there is evidence indicating that programmers varied in their degree of semantic sensitivity that may have been washed out by grand mean averages. For example, plots of individual changes in ERP amplitudes between well-formed code and semantically implausible code show that participants varied in the *degree* of amplitude change as well as the *direction*; this was the case in whole group analyses (Fig. 7), between-group analyses on programming expertise (Fig. 10 and 12), and between-group analyses on English proficiency (Fig. 16 and 18). Therefore, while grand mean ERPs suggest that N400 effects to meaning violations were similar between Experts/Novices and English-High/English-

Moderate individuals, it is possible that semantic sensitivity influences code comprehension on an individual basis.

Further support for this idea comes from the fact that participants varied in regards to the *difference* in their structural validity *d'* measures (structural validity *d'* when code is semantically plausible minus structural validity *d'* when code is semantically implausible) (Fig. 29). A negative change in *d'* would indicate a larger *d'* score when code was semantically implausible, with consistent English semantics possibly swaying individuals to judge structurally invalid code as acceptable. Meanwhile, a positive change in *d'* would indicate a larger *d'* score when code was semantically plausible, with inconsistent English semantics possibly swaying individuals to judge structurally valid code as unacceptable. The variability across all

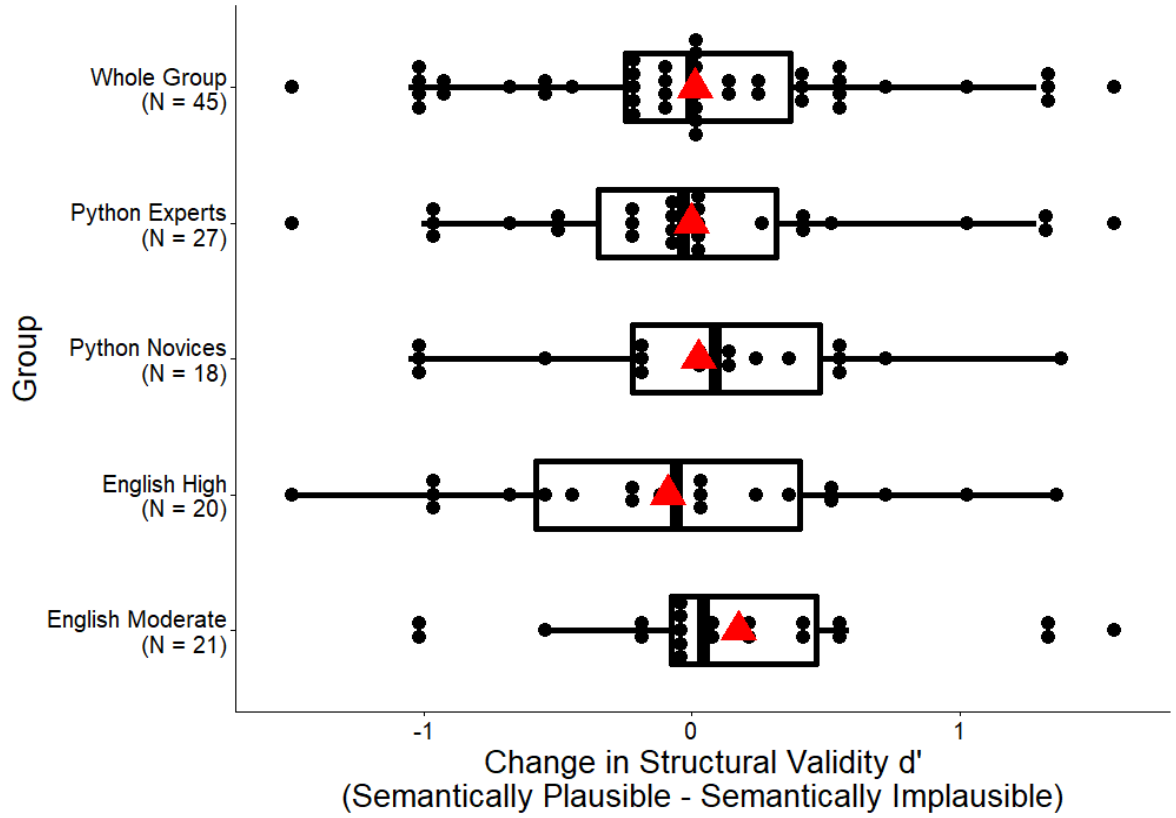

Fig. 29. Change in *d'* scores (Structural Validity *d'* when code is semantically plausible minus Structural Validity *d'* when code is semantically implausible). Triangle denotes the mean.

participants, regardless of programming expertise or English proficiency, implies that sensitivity to semantics is unique to individuals, and the near-zero average change in *d'* is not reflective of typical performance.

In general, semantic sensitivity could be indicative of cognitive style, such as the degree to which an individual relies on general semantic processing or top-down world knowledge schemas to interpret incoming stimuli of all types. As such, semantic sensitivity may be a trait that influences an individual's mental model construction across all domains, not merely in programming contexts. This would be consistent with the finding that mathematical thinking is not independent of semantic knowledge for modelers who rely on schemas that dictate the relationship between operators and objects (Guthormsen et al., 2015). Additionally, native speakers of natural languages vary in the degree to which they exhibit N400 effects to semantic anomalies and P600 effects to syntax anomalies (Tanner et al., 2014). N400/P600 dominance is a spectrum, with some individuals being more prone to exhibiting global N400 effects to all violations, and others more prone to exhibiting global P600 effects. To fully capture the relationship between meaning and form during code comprehension, it would be beneficial to investigate semantic sensitivity and structural sensitivity as a continuum in programmers, which could provide a more complete picture of their joint influence on mental model creation.

4.5. General Discussion and Implications

Across domains, humans have developed expectations for how stimuli should occur based on their expertise in the area, including natural language (Osterhout & Nicol, 1999), mathematics (Guthormsen et al., 2015), and music (Besson & Macar, 1987; Calma-Roddin & Drury, 2020; Janata, 1995; Patel et al., 1998). As a result, the integration of meaning and form is central to comprehension across domains. The symbols and rules may change from domain to

domain, but affixing meaning to symbols is what allows us to evaluate them and, if necessary, employ the relevant resources to reinterpret them in cases where what we see or hear does not match our internal representations. Highly proficient individuals develop schemas that incorporate more information about structure, leading them to build mental models centered around form with meaning as an added guide (Staggers & Norcio, 1993). On the contrary, moderate to low proficiency individuals have yet to obtain a deep understanding of conceptual knowledge in the domain, leading them to process stimulus by relying on lower-level semantic features or general world knowledge schemas (Johnson-Laird, 1983; 2004).

This dissertation presents the first evidence that humans also rely on the integration of meaning and form to understand programming languages. Programmers respond with different neural signatures to meaning and form in code that mirror patterns found in natural language, mathematics, and music. The reported results also support prior research suggesting that programming expertise mediates the joint influence that meaning and form exert on mental model creation during code comprehension (Mayer, 1981; Robins et al., 2013). Although Experts and Novices displayed similar neural responses to semantic anomalies, Experts exhibited more prominent neural responses to structural anomalies that indicate a stronger sensitivity to form compared to novices. Therefore, while Experts' deep syntax knowledge is strong enough to override cases in which inconsistent meaning schemas may interfere with their mental model construction, Novices have more shallow understandings of syntax and rely on their meaning schemas to build their mental models.

If we wish to improve programming education, giving novices the correct foundational knowledge to support the development of their mental models of programming in accurate and efficient ways would be more effective than merely teaching them the declarative knowledge and

syntax rules (Qian & Lehman, 2017). Now that we have established how expert programmers respond to violations of meaning and form during code comprehension, measuring learners' ERPs to similar violations may be one method to diagnose their struggles. If an individual judges well-formed and anomalous code at similar acceptability rates as experts, but displays novice-like brain responses, they may simply need additional practice with syntax rules before they incorporate them into their schemas. On the other hand, if an individual judges code at similar acceptability rates as novices, but displays native-like responses, they may have developed erroneous schemas that have led them to misrepresent programming concepts and rules that are fundamentally false. This would require a correction of an individual's acquired programming *content*. Combining behavioral and brain responses to the same stimulus allows us to tap into somebody's knowledge representation in two dimensions, providing insight into their stage of learning and the stability of their mental model.

4.6. Limitations

The present study contains several limitations relating to individual differences that should be addressed in future studies. To start, the present study analyzed usable data from 45 participants, which is effective for studying group level effects, but limits takeaways regarding individual differences. Programming ability draws upon each individual's rich history, including their natural language background, mathematics ability, and general cognitive abilities (e.g., working memory and fluid reasoning), which warrants attention if the goal is to understand "the difficulties and complexities faced by the students" (Jenkins, 2002). The best way to do so is to retain what makes every individual unique and move away from the dichotomization of these factors. As such, even though the reported results suggest that code comprehension differs between expert and novice programmers (with English proficiency trending in a similar

direction, albeit non-significant), employing group splits to represent programming expertise and English proficiency removes the variability that is inherent in the population we wish to study. Instead, increasing recruitment to larger sample sizes will allow for a deeper dive into the individual differences that contribute to mental model construction during code comprehension, which can be conducted via methods such as correlations and linear regression. Doing so would also mitigate confounds that arise as a result of group splits, such as the restricted range of English reading comprehension ability that characterized the present study's participants, as well as the overlap of English proficiency with programming expertise.

4.7. Future Directions

The implication that programmers rely on schemas and expertise to integrate meaning and form during code comprehension opens up additional avenues to explore the intricacies of programming, including, but not limited to: broadening the scope of semantic context, capturing the progression of programming expertise, relating the effects of bilingualism to multiple programming languages, and investigating the generalizability of these findings with respect to other programming languages.

4.7.1. Redefining semantic context.

4.7.1. Manipulating semantic context on a local level. In this study, semantically implausible code was created by replacing thematically consistent variable names with thematically inconsistent variables names. However, the effect of semantic context with regards to variables can be further tested by using pseudo-words or letter strings as variable names. If programmers exhibit stronger N400 amplitudes to semantically implausible variable names than to pseudo-words/letter strings, then the N400 effect would be truly reflective of the degree to which English semantics matches the expected semantic context of the code. On the contrary, if

programmers exhibit the same N400 amplitude to pseudo-words/letter strings and semantically implausible variable names, it can be argued that it is not the well- or ill-formed English semantics specifically that impacts processing, but rather the mere presence of unexpected stimulus.

Another method is to manipulate the relationship that the global variable-iterator variable pair has with the action that is carried out by the line of code. Specifically, the global variable and iterator variable could be semantically related words, but their association is incompatible with the items that are stored in the global variable. For example, consider the list *pets = ["dog", "cat", "hamster"]*; a well-formed line of code would be *for animal in pets: print(animal)*. Rather than using a semantically unrelated word in place of *animal* (e.g., *fruit*), we can use a word like *owner*. Although pets typically have owners, it would not make thematic sense to use *owner* to iterate through the list *pets*, which contains examples of animals that could be pets. As a result, the line of code *for owner in pets: print(owner)* could elicit stronger N400 effects than the well-formed counterpart due to the semantic mismatch during the retrieval process. However, a scenario like this could also elicit stronger P600 effects than the well-formed counterpart due to the word *owner* fulfilling an unexpected role (i.e., it is not an example of an animal that could be a pet but rather a human that can possess a pet); this would be similar to the "semantic attraction" phenomenon in natural language in which the thematic role of a verb as implied by the provided context is inconsistent with the role that it usually inhabits (Kim & Osterhout, 2005; Kuperberg, 2007). Depending on the results, such a manipulation could provide further insight into how meaning interacts with form to influence programmers' construction of mental models during code comprehension, as well as the type of semantic world knowledge schemas that may guide top-down processing.

4.7.1. Manipulating semantic context on a global level. Semantic context does not solely matter in single-line situations; rather, code often occurs in organized chunks, similar to how sentences make up large passages in natural languages. One of the hallmarks of an efficient program lies in the continuity from line to line; for example, whether the existence of a defined variable or line of code makes sense relative to the surrounding code, as well as the overarching goal of the program. As such, programmers may evaluate the same line of code differently depending on whether they are considering it from an isolated standpoint or within the context of a larger chunk of code. In English, this would be akin to asking if the sentence "The cat won't eat the food" is acceptable by itself, versus if it were to appear in the middle of a paragraph detailing how to plant a tree.

Another method of manipulating global semantic context is to provide participants with a title or a comment that may or may not reflect the true output of an upcoming chunk of code (e.g., "This program calculates a person's tax return."). In doing so, participants would retrieve the appropriate schema (e.g., the common steps associated with calculating taxes) and attempt to integrate it with the syntax to construct a mental model of the program's ultimate goal. If consistent titles aid processing while inconsistent titles hinders it, this would suggest that world knowledge schemas influence code comprehension on a global level, similar to Bransford and Johnson's (1972) study on the facilitating effects that appropriate world knowledge schemas have on readers' recall of a passage. Although the ERP method is constrained in terms of its scope, other methods such as eye-tracking or fMRI could be employed to study how semantic context influences programmers' understanding of larger chunks of code.

4.7.2. Effect of task demands. As alluded to previously, the variability in the predictors and neural correlates that have been associated with programming aptitude may partially be due

to the task demands (e.g., Castelhano et al., 2021; Liu et al., 2020). In the present study, participants were asked to determine the acceptability of code, but not their output or their overall goal. As such, they may not have engaged as deeply with the semantics or their relationship with the code structure. Furthermore, the repetition of trial types may have clued participants into which errors to look for, allowing them to focus on individual parts code rather than the entire line as a whole. To address this, an additional prompt that instructs participants to report or select the correct output and/or goal of every trial following their acceptability judgment would provide more information on whether code *comprehension* truly took place. Alternatively, participants could be asked to complete the same study with two different task types: one in which they provide acceptability judgments, and one in which they provide the correct output or goal. Participants' ERPs for the two task types could be compared to each other to determine how mental model construction may differ as a function of engagement with the code. This modification could also shed light on whether novices can be encouraged to think "more like" an expert when deliberately encouraged to. If so, this would indicate that novices' knowledge representations of code are malleable, and that similar methods could be used to promote higher-level understanding of programming concepts.

4.7.3. Evolution of programming expertise. There have been many studies indicating that second language learners' processing of syntax rules becomes more native-like with increasing proficiency (McLaughlin et al., 2010). The present study established that programmers differed in their code acceptability judgments and brain responses to structural violations based on expertise, but individual participants varied in the amount of experience they had with using Python and how they learned it. A longitudinal study that follows learners through the same number of years of Python instruction would control for Python experience

within the sample while showing how brain responses change as individuals transition from novices to experts. Changes in code acceptability judgments and ERPs with increased proficiency would provide evidence that a programmer has developed the ability to integrate meaning schemas with deep structural knowledge to construct their mental model. Depending on the results, it is possible that programming expertise could be diagnosed via three methods: traditional knowledge assessments (e.g., multiple-choice tests or coding/debugging tasks), acceptability judgments, and their neural signatures to violations of form.

4.7.4. Effect of programming "bilingualism." Prior research on second language learning suggests that bilingualism aids in faster acquisition of a third language, often above and beyond the time it takes monolinguals to learn a second language (Bartolotti & Marian, 2012; Eisenstein, 1980; Lambert, 1981). Language similarity plays a key role, but the general added experience of having already learned a second language provides a larger lexical database for learners to draw upon, allowing for rapid internalization of new vocabulary and syntax rules (Eisenstein, 1980; Lambert, 1981). Previous research has shown that knowing a programming language can provide facilitating and negative transfer effects in acquiring additional programming languages (Scholtz & Wiedenbeck, 1990; Wu & Anderson, 1990). To investigate these effects on programmers' mental models, future research comparing the online processing of Python meaning and form between Python-only users and users with high proficiency in a different programming language may provide insight regarding potential benefits of "bilingualism" in programming contexts.

4.7.5. Generalizability to other programming languages. Finally, Python was selected for this study due to its simplicity in form and similarity to the English language. However, an important question arises regarding the generalizability of the current findings to other

programming languages that may have more complex syntax structures, a heavier reliance on symbols, and a steeper learning curve. Furthermore, some languages were designed to be more all-purpose than others due to their individual specializations, resulting in a variety of language types that support different programming styles (e.g., procedural, functional, object-oriented, etc.). Therefore, even if the processing of meaning and form is fundamental to all coding environments, the emphasis on one over the other, as well as how each type of information is processed, may differ across programming languages. Future investigations that compare and contrast programmers' brain responses to different programming languages would provide a more complete picture of whether the processing of meaning and form—as well as the similarity of their neural signatures to natural languages—is limited to Python or representative of all programming languages. Furthermore, this may indicate whether programmers develop a single mental model that encompasses all programming languages, or individual mental models for separate programming languages.

4.8. Conclusions

The present study marked the first step in understanding how meaning and form are integrated by programmers to construct mental models of a program's ultimate goal. Specifically, programmers' distinct brain responses to semantic violations and structural violations suggest that meaning and form are distinct sources of information that jointly influence code comprehension, similar to other symbolic systems such as natural language and mathematics. Programming expertise is associated with a stronger sensitivity to structural cues, suggesting that more proficient programmers focus on form during code comprehension, whereas less proficient programmers rely on world knowledge meaning schemas to understand and evaluate code. Follow-up analyses revealed that semantic sensitivity may be a trait that

characterizes the extent to which individuals prioritize meaning over form across domains. Meanwhile, English proficiency may influence code comprehension in a similar fashion as programming expertise, but the results were weak and contained potential confounds that should be addressed more systematically. Future studies expanding the scope to other programming languages should be carried out to determine if the current results reflect programmers' construction of mental models encompassing all programming languages, or a process tailored for Python comprehension.

References

Anderson, R.C. (1978). Schema-Directed Processes in Language Comprehension. In: Lesgold, A.M., Pellegrino, J.W., Fokkema, S.D., Glaser, R. (eds) Cognitive Psychology and Instruction. Nato Conference Series, vol 5. Springer, Boston, MA. https://doi.org/10.1007/978-1-4684-2535-2_8

Anderson, J. R., & Jeffries, R. (1985). Novice LISP errors: Undetected losses of information from working memory. *Human–Computer Interaction, 1*(2), 107-131. http://dx.doi.org/10.1207/s15327051hci0102_2

Alspaugh, C. A. (1972). Identification of some components of computer programming aptitude. *Journal for Research in Mathematics Education, 3*(2), 89-98.

Aurnhammer, C., Delogu, F., Schulz, M., Brouwer, H., & Crocker, M. W. (2021). Retrieval (N400) and integration (P600) in expectation-based comprehension. *PLoS One, 16*(9), e0257430.

Aurnhammer, C., Delogu, F., Brouwer, H., & Crocker, M. W. (2023). The P600 as a continuous index of integration effort. *Psychophysiology,* e14302.

Austin, H. S. (1987, February). Predictors of pascal programming achievement for community college students. In *Proceedings of the Eighteenth SIGCSE Technical Symposium on Computer Science Education* (pp. 161-164).

Ausubel, D. P. (1960). The use of advance organizers in the learning and retention of meaningful verbal material. *Journal of Educational Psychology, 51*(5), 267.

Barnett, S. M., & Ceci, S. J. (2002). When and where do we apply what we learn?: A taxonomy for far transfer. *Psychological Bulletin, 128*(4), 612.

Barnitz, J. G. (1986). Toward understanding the effects of cross-cultural schemata and discourse

structure on second language reading comprehension. *Journal of Reading Behavior, 18*(2), 95-116.

Bartolotti, J., & Marian, V. (2012). Language learning and control in monolinguals and bilinguals. *Cognitive Science, 36*(6), 1129-1147.

Basili, V., Tesoriero, R., Costa, P., Lindvall, M., Rus, I., Shull, F., & Zelkowitz, M. (2001). Building an Experience Base for Software Engineering: A report on the first CeBASE eWorkshop. In *Product Focused Software Process Improvement: Third International Conference, PROFES 2001 Kaiserslautern, Germany, September 10–13, 2001 Proceedings 3* (pp. 110-125). Springer Berlin Heidelberg.

Bassok, M., Chase, V., & Martin, S. (1998). Adding apples and oranges: Alignment of semantic and formal knowledge. *Cognitive Psychology, 35,* 99-134.

Bayman, P., & Mayer, R. E. (1983). A diagnosis of beginning programmers' misconceptions of BASIC programming statements. *Communications of the ACM, 26*(9), 677-679.

Bayman, P., & Mayer, R. E. (1988). Using conceptual models to teach BASIC computer programming. *Journal of Educational Psychology, 80*(3), 291.

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, *39*(2), 32–36. https://dl.acm.org/doi/10.1145/1272848.1272879

Bennedsen, J., & Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM Inroads*, *10*(2), 30–36. https://doi.org/10.1145/3324888

Bentin, S., Kutas, M., & Hillyard, S. A. (1993). Electrophysiological evidence for task effects on semantic priming in auditory word processing. *Psychophysiology, 30*, 161-169.

Besson, M., & Macar, F. (1987). An event-related potential analysis of incongruity in music and other non-linguistic contexts. *Psychophysiology, 24*(1), 14-25.

Bishop-Clark, C. (1995). Cognitive style, personality, and computer programming. *Computers in Human Behavior, 11*(2), 241-260.

Bonar, J., & Soloway, E. (1985). Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human–Computer Interaction, 1*(2), 133-161. http://dx.doi.org/ 10.1207/s15327051hci0102_3

Bornkessel-Schlesewsky, I., & Schlesewsky, M. (2013). Reconciling time, space and function: a new dorsal–ventral stream model of sentence comprehension. *Brain and Language, 125*(1), 60-76.

Bransford, J. (1979). Human Cognition: Learning, Understanding, and Remembering. Belmont, CA: Wadsworth.

Bransford, J. D., & Johnson, M. K. (1972). Contextual prerequisites for understanding: Some investigations of comprehension and recall. *Journal of Verbal Learning and Verbal Behavior, 11*(6), 717-726.

Brooks, R.E. (1977). Towards a theory of the cognitive processes in computer programming. *International Journal of Man-Machine Studies, 9*, 737–751.

Brooks, R.E. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies, 18*, 543–554

Brown, J. I. (1960). The Nelson-Denny Reading Test. Houghton Mifflin.

Bruckman, A., & Edwards, E. (1999, May). Should we leverage natural-language knowledge? An analysis of user errors in a natural-language-style programming language. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (pp. 207-214). http://dx.doi.org/10.1145/302979.303040

Buse, R. P., & Zimmermann, T. (2012, June). Information needs for software development analytics. In *2012 34th International Conference on Software Engineering (ICSE)* (pp. 987-996). IEEE.

Butcher, D. F., & Muth, W. A. (1985). Predicting performance in an introductory computer science course. *Communications of the ACM, 28*(3), 263-268.

Calma-Roddin, N., & Drury, J. E. (2020). Music, language, and the N400: ERP interference patterns across cognitive domains. *Scientific Reports, 10*(1), 1-14.

Camblin, C. C., Ledoux, K., Boudewyn, M., Gordon, P. C., & Swaab, T. Y. (2007). Processing new and repeated names: Effects of coreference on repetition priming with speech and fast RSVP. *Brain Research, 1146*, 172-184.

Canonne, C., & Aucouturier, J. J. (2016). Play together, think alike: Shared mental models in expert music improvisers. *Psychology of Music, 44*(3), 544-558.

Carrell, P. L. (1984). Evidence of a formal schema in second language comprehension. *Language Learning, 34*(2), 87-108.

Carroll, J. M. (1984). MINIMALIST TRAINING. *Datamation*, *30*(18), 5p.

Castelhano, J., Duarte, I. C., Duraes, J., Madeira, H., & Castelo-Branco, M. (2021). Reading and calculation neural systems and their weighted adaptive use for programming skills. *Neural Plasticity, 2021*.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology, 4*(1), 55-81.

CHESScom. (2022, September 15). *The Best Chess Openings For Beginners.* Chess.com. Retrieved from https://www.chess.com/article/view/the-best-chess-openings-for-beginners

Clancy, M. (2005). Misconceptions and attitudes that interfere with learning to program. In *Computer Science Education Research* (pp. 95-110). Taylor & Francis.

Codecademy (n.d.). *Codecademy.* Retrieved from https://www.codecademy.com/

Collins, A., & Gentner, D. (1987). How people construct mental models. *Cultural Models in Language and Thought, 243*(1987), 243-265.

Cooper, G., & Sweller, J. (1987). Effects of schema acquisition and rule automation on mathematical problem-solving transfer. *Journal of Educational Psychology, 79*(4), 347–362. https://doi.org/10.1037/0022-0663.79.4.347

Curtis, B. (1984, March). Fifteen years of psychology in software engineering: Individual differences and cognitive science. In *Proceedings of the 7th International Conference on Software Engineering* (pp. 97-106).

Dalbey, J., & Linn, M. C. (1985). The demands and requirements of computer programming: a literature review. *Journal of Educational Computing Research*, *1*(3), 253–274. https://doi.org/10.2190/BC76-8479-YM0X-7FUA

Delogu, F., Brouwer, H., & Crocker, M. W. (2019). Event-related potentials index lexical retrieval (N400) and integration (P600) during language comprehension. *Brain and Cognition, 135*, 103569.

Diachek, E., Blank, I., Siegelman, M., Affourtit, J., & Fedorenko, E. (2020). The domain-general multiple demand (MD) network does not support core aspects of language comprehension: a large-scale fMRI investigation. *Journal of Neuroscience, 40*(23), 4536-4550.

Ditman, T., Holcomb, P. J., & Kuperberg, G. R. (2007). An investigation of concurrent ERP and self-paced reading methodologies. *Psychophysiology, 44*(6), 927-935.

Doukakis, D., Grigoriadou, M., & Tsaganou, G. (2007). Understanding the programming variable concept with animated interactive analogies. In *Proceedings of the The 8th Hellenic European Research on Computer Mathematics & Its Applications Conference (HERCMA'07).*

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research, 2*(1), 57-73.

Eisenstein, M. (1980). Childhood bilingualism and adult language learning aptitude. *International Review of Applied Psychology.*

Elshoff, J. L., & Marcotty, M. (1982). Improving computer program readability to aid modification. *Communications of the ACM, 25*(8), 512-521.

Erodogan, Y., Aydin, E., & Kabaca, T. (2008). Exploring the psychological predictors of programming achievement. *Journal of Instructional Psychology, 35*(3), 264.

Fedorenko, E., Ivanova, A., Dhamala, R., & Bers, M. U. (2019). The language of programming: A cognitive perspective. *Trends in Cognitive Sciences*, *23*(7), 525-528.

Fedorenko, E., & Thompson-Schill, S. L. (2014). Reworking the language network. *Trends in Cognitive Sciences, 18*(3), 120-126.

Fensham, P. J., & West, L. H. T. (1976). Prior knowledge or advance organizers as effective variables in chemical learning. *Journal of Research in Science Teaching, 13*(4), 297-306.

Ferreira, F., Bailey, K. G., & Ferraro, V. (2002). Good-enough representations in language comprehension. *Current Directions in Psychological Science, 11*(1), 11-15.

Fleury, A. E. (1991). Parameter passing: The rules the students construct. *ACM SIGCSE Bulletin, 23*(1), 283-286. https://dl.acm.org/doi/10.1145/107005.107066

Floyd, B., Santander, T., & Weimer, W. (2017, May). Decoding the representation of code in the

brain: An fMRI study of code review and expertise. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)* (pp. 175-186). IEEE.

Foucart, A., & Frenck-Mestre, C. (2011). Grammatical gender processing in L2: Electrophysiological evidence of the effect of L1–L2 syntactic similarity. *Bilingualism: Language and Cognition, 14*(3), 379-399.

Foucart, A., & Frenck-Mestre, C. (2012). Can late L2 learners acquire new grammatical features? Evidence from ERPs and eye-tracking. *Journal of Memory and Language, 66*(1), 226-248.

Friederici, A. D., Hahne, A., & Mecklinger, A. (1996). Temporal structure of syntactic parsing: early and late event-related brain potential effects. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 22*(5), 1219.

Friederici, A. D., Pfeifer, E., & Hahne, A. (1993). Event-related brain potentials during natural speech processing: Effects of semantic, morphological and syntactic violations. *Cognitive Brain Research, 1*(3), 183-192.

Galfano, G., Mazza, V., Angrilli, A., & Umiltà, C. (2004). Electrophysiological correlates of stimulus-driven multiplication facts retrieval. *Neuropsychologia, 42*(10), 1370-1382.

Gentner, D., & Medina, J. (1998). Similarity and the development of rules. *Cognition, 65*(2-3), 263-297.

Gentner, D., & Stevens, A. L. (Eds.). (1983). *Mental Models* (1st ed.). Psychology Press. https://doi.org/10.4324/9781315802725

Gentner, D., & Stevens, A. L. (Eds.). (2014). *Mental models*. Psychology Press.

Ghosh, V. E., & Gilboa, A. (2014). What is a memory schema? A historical perspective on current neuroscience literature. *Neuropsychologia, 53*, 104-114.

Gobet, F., & Simon, H. A. (1998). Expert chess memory: Revisiting the chunking hypothesis. *Memory, 6*(3), 225-255.

Greca, I. M., & Moreira, M. A. (2000). Mental models, conceptual models, and modelling. *International Journal of Science Education, 22*(1), 1-11.

Green, T. R. G. (1977). Conditional program statements and their comprehensibility to professional programmers. *Journal of Occupational Psychology, 50*(2), 93-109. http://dx.doi.org/10.1111/j.2044-8325.1977.tb00363.x

Guthormsen, A. M., Fisher, K. J., Bassok, M., Osterhout, L., DeWolf, M., & Holyoak, K. J. (2016). Conceptual integration of arithmetic operations with real-world knowledge: Evidence from event-related potentials. *Cognitive Science, 40*(3), 723-757.

Hagoort, P. (2005). On Broca, brain, and binding: a new framework. *Trends in Cognitive Sciences, 9*(9), 416-423.

Hagoort, P., Brown, C., & Groothusen, J. (1993). The syntactic positive shift (SPS) as an ERP measure of syntactic processing. *Language and Cognitive Processes, 8*(4), 439-483.

Haneef, N. J. (1998). Software documentation and readability: a proposed process improvement. *ACM SIGSOFT Software Engineering Notes, 23*(3), 75-77.

Hansen, C. (n.d.). *Python Keywords: An Introduction.* Real Python. Retrieved from https://realpython.com/python-keywords/#python-keywords

Hoff, E. (2013). *Language Development.* Cengage Learning.

Hofstadter, D. R. (2001). Analogy as the core of cognition. *The analogical mind: Perspectives from cognitive science*, 499-538.

Holcomb, P. J. (1988). Automatic and attentional processing: An event-related brain potential analysis of semantic priming. *Brain and Language, 35*, 66-85.

Holcomb, P. J., & Neville, H. J. (1990). Auditory and visual semantic priming in lexical decision: A comparison using event-related brain potentials. *Language and Cognitive Processes, 5*, 281-312.

Ivanova, A. A., Srikant, S., Sueoka, Y., Kean, H. H., Dhamala, R., O'Reilly, U. M., ... & Fedorenko, E. (2020). Comprehension of computer code relies primarily on domain-general executive brain regions. *elife*, *9*, e58906.

Janata, P. (1995). ERP measures assay the degree of expectancy violation of harmonic contexts in music. *Journal of Cognitive Neuroscience, 7*(2), 153-164.

Jasper, H. H. (1958). Ten-twenty electrode system of the international federation. *Electroencephalography and Clinical Neurophysiology, 10*, 371-375.

Jenkins, T. (2002, August). On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences* (Vol. 4, No. 2002, pp. 53-58).

Johnson-Laird, P. N. (1983). *Mental models: Towards a cognitive science of language, inference,*

and consciousness (No. 6). Harvard University Press.

Johnson-Laird, P. N. (2004). The history of mental models. In *Psychology of Reasoning* (pp. 189-222). Psychology Press.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., & Herman, G. L. (2010, March). Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (pp. 107-111). http://dx.doi.org/10.1145/1734263.1734299

Kagan, D. M., & Douthat, J. M. (1985). Personality and learning FORTRAN. *International*

Journal of Man-Machine Studies, 22(4), 395-402.

Kahney, H. (1983, December). What do novice programmers know about recursion. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (pp. 235-239).

Karimi, H., & Ferreira, F. (2016). Good-enough linguistic representations and online cognitive equilibrium in language processing. *Quarterly Journal of Experimental Psychology, 69*(5), 1013-1040.

Kim, A., & Osterhout, L. (2005). The independence of combinatory semantic processing: Evidence from event-related potentials. *Journal of Memory and Language, 52*(2), 205-225.

Knight, J. C., & Myers, E. A. (1993). An improved inspection technique. *Communications of the ACM, 36*(11), 50-61.

Knuth, D. E. (1984). Literate programming. *The Computer Journal, 27*(2), 97-111.

Konvalina, J., Wileman, S. A., & Stephens, L. J. (1983). Math proficiency: A key to success for computer science students. *Communications of the ACM, 26*(5), 377-382.

Krumhansl, C. L., & Castellano, M. A. (1983). Dynamic processes in music perception. *Memory & Cognition, 11*(4), 325-334.

Kuo, C. H., Mottarella, M., Haile, T., & Prat, C. S. (2022, September). Predicting programming success: How intermittent knowledge assessments, individual psychometrics, and resting-state EEG predict Python programming and debugging skills. In *2022 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)* (pp. 1-6). IEEE.

Kuperberg, G. R. (2007). Neural mechanisms of language comprehension: Challenges to syntax.

Brain Research, 1146, 23-49.

Kurtz, B. L. (1980, February). Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. In *Proceedings of the Eleventh SIGCSE Technical Symposium on Computer Science Education* (pp. 110-117).

Kutas, M., & Federmeier, K. D. (2011). Thirty years and counting: finding meaning in the N400 component of the event-related brain potential (ERP). *Annual Review of Psychology, 62,* 621-647.

Kutas, M., Federmeier, K. D., & Urbach, T. P. (2014). The "negatives" and "positives" of prediction in language. In M. S. Gazzaniga & G. R. Mangun (Eds.), *The cognitive neurosciences* (pp. 649–656). Boston Review.

Kutas, M., & Hillyard, S. A. (1980). Reading senseless sentences: Brain potentials reflect semantic incongruity. *Science, 207*(4427), 203-205.

Kutas, M., Van Petten, C. K., & Kluender, R. (2006). Psycholinguistics electrified II (1994–2005). In Handbook of Psycholinguistics (pp. 659-724). Academic Press.

Lambert, W. E. (1981). Bilingualism and language acquisition. *Annals of the New York Academy of Sciences, 379*, 9–22. https://doi.org/10.1111/j.1749-6632.1981.tb41993.x

Lane, D. M., & Chang, Y. H. A. (2018). Chess knowledge predicts chess memory even after controlling for chess experience: Evidence for the role of high-level processes. *Memory & Cognition, 46*, 337-348.

Lau, E. F., Phillips, C., & Poeppel, D. (2008). A cortical network for semantics:(de) constructing the N400. *Nature Reviews Neuroscience, 9*(12), 920-933.

Ledoux, K., Gordon, P. C., Camblin, C. C., & Swaab, T. Y. (2007). Coreference and lexical repetition: Mechanisms of discourse integration. *Memory & Cognition, 35*, 801-815.

Leman, M. (2012). Music and schema theory: Cognitive foundations of systematic musicology (Vol. 31). Springer Science & Business Media.

Lesh, R. A. (1976). The influence of an advanced organizer on two types of instructional units about finite geometries. *Journal for Research in Mathematics Education, 7*(2), 82-86.

Lewis, A. G., Schoffelen, J. M., Schriefers, H., & Bastiaansen, M. (2016). A predictive coding perspective on beta oscillations during sentence-level language comprehension. *Frontiers in Human Neuroscience, 10*, 85.

Lipshitz, R., & Shaul, O. B. (1997). Schemata and mental models in recognition-primed decision making. *Naturalistic Decision Making*, 293-303.

Liu, Y. F., Kim, J., Wilson, C., & Bedny, M. (2020). Computer code comprehension shares neural resources with formal logical inference in the fronto-parietal network. *elife, 9*, e59340.

Luck, S. J. (2014). *An introduction to the event-related potential technique*. MIT press.

Ma, L. (2007). *Investigating and improving novice programmers' mental models of programming concepts* (Doctoral dissertation, University of Strathclyde).

Marian, V., Blumenfeld, H. K., & Kaushanskaya, M. (2007). The Language Experience and Proficiency Questionnaire (LEAP-Q): Assessing language profiles in bilinguals and multilinguals.

Mayer, R. E. (1975). Different problem-solving competencies established in learning computer programming with and without meaningful models. *Journal of Educational Psychology, 67*(6), 725.

Mayer, R. E. (1976). Some conditions of meaningful learning for computer programming: Advance organizers and subject control of frame order. *Journal of Educational*

Psychology, 68(2), 143.

Mayer, R. E. (1980). Elaboration techniques for technical text: An experimental test of the learning strategy hypothesis. *Journal of Educational Psychology, 72*, 770-784.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. *ACM Computing Surveys (CSUR)*, *13*(1), 121-141.

McCarthy, D. (1946). Language development in children. In L. Carmichael (Ed.), Manual of Child Psychology (pp. 476–581). John Wiley & Sons, Inc. https://doi.org/10.1037/10756-010

McLaughlin, J., Osterhout, L., & Kim, A. (2004). Neural correlates of second-language word learning: Minimal instruction produces rapid change. *Nature Neuroscience, 7*(7), 703-704.

McLaughlin, J., Tanner, D., Pitkänen, I., Frenck-Mestre, C., Inoue, K., Valentine, G., & Osterhout, L. (2010). Brain potentials reveal discrete stages of L2 grammatical learning. *Language Learning, 60*, 123-150.

McVee, M. B., Dunsmore, K., & Gavelek, J. R. (2005). Schema theory revisited. *Review of Educational Research, 75*(4), 531-566.

Mertala, P. (2019). Young children's conceptions of computers, code, and the Internet. *International Journal of Child-Computer Interaction, 19*, 56-66.

Meyer, D. M., & Schvaneveldt, R. W. (1971). Facilitation in recognizing pairs of words: Evidence of a dependence between retrieval operations. *Journal of Experimental Psychology, 90*, 227-234.

Miller, C. S. (2014). Metonymy and reference-point errors in novice programming. *Computer Science Education, 24*(2-3), 123-152.

Milojkovic, J. D. (1982). Chess imagery in novice and master. *Journal of Mental Imagery, 6*(2), 125–144.

Mladenović, M., Boljat, I., & Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies, 23*, 1483-1500.

Muller, O., Ginat, D., & Haberman, B. (2007, June). Pattern-oriented instruction and its influence on problem decomposition and solution construction. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 151-155).

National Education Policy (2020). Ministry of Human Resource Development. Government of India. Retrieved from https://www.education.gov.in/sites/upload_files/mhrd/files/NEP_Final_English_0.pdf

Neely, J. H. (1991). Semantic priming effects in visual word recognition: A selective review of current findings and theories. In D. Besner & G. Humphreys (Eds.), *Basic Processes in Reading: Visual Word Recognition* (pp. 264-336). Hillsdale, NJ: Erlbaum.

Neville, H., Nicol, J. L., Barss, A., Forster, K. I., & Garrett, M. F. (1991). Syntactically based sentence processing classes: Evidence from event-related brain potentials. *Journal of Cognitive Neuroscience, 3*(2), 151-165.

Nichols, A. L. (2021, May 18). *Coding could count for foreign language credit under bill*. AP News. Retrieved from https://apnews.com/article/technology-bills-business-education-government-and-politics-27d66b48eb1cded44c2fc10bb8376c9e

Niedeggen, M., & Rösler, F. (1999). N400 effects reflect activation spread during retrieval of arithmetic facts. *Psychological Science, 10*(3), 271-276.

Norman, D. A. (2014). Some observations on mental models. In Mental Models (pp. 15-22). Psychology Press.

Núñez-Peña, M. I., & Honrubia-Serrano, M. L. (2004). P600 related to rule violation in an arithmetic task. *Cognitive Brain Research, 18*(2), 130-141.

Osterhout, L., & Holcomb, P. J. (1992). Event-related brain potentials elicited by syntactic anomaly. *Journal of Memory and Language, 31*(6), 785-806.

Osterhout, L., & Nicol, J. (1999). On the distinctiveness, independence, and time course of the brain responses to syntactic and semantic anomalies. *Language and Cognitive Processes, 14*(3), 283-317.

Owen, E., & Sweller, J. (1985). What do students learn while solving mathematics problems?. *Journal of Educational Psychology, 77*(3), 272.

Pallier, C., Devauchelle, A. D., & Dehaene, S. (2011). Cortical representation of the constituent structure of sentences. *Proceedings of the National Academy of Sciences, 108*(6), 2522-2527.

Patel, A. D., Gibson, E., Ratner, J., Besson, M., & Holcomb, P. J. (1998). Processing syntactic relations in language and music: An event-related potential study. *Journal of Cognitive Neuroscience, 10*(6), 717-733.

Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research, 2*(1), 25-36. http://dx.doi.org/10.2190/689T-1R2A-X4W4-29J2

Pea, R. D., & Kurland, D. M. (1983). On the Cognitive Prerequisites of Learning Computer Programming. Technical Report No. 18.

Petersen, C. G., & Howe, T. G. (1979). Predicting academic success in introduction to

computers. *AEDS Journal*, *12*(4), 182-191.

Pocius, K. E. (1991). Personality factors in human-computer interaction: A review of the literature. *Computers in Human Behavior, 7*(3), 103-135.

Prat, C. S., Madhyastha, T. M., Mottarella, M. J., & Kuo, C. H. (2020). Relating natural language aptitude to individual differences in learning programming languages. *Scientific Reports, 10*(1), 3817.

Prat, C. S., Yamasaki, B. L., Kluender, R. A., & Stocco, A. (2016). Resting-state qEEG predicts rate of second language learning in adults. *Brain and Language, 157*, 44-50.

Prat, C. S., Yamasaki, B. L., & Peterson, E. R. (2019). Individual differences in resting-state brain rhythms uniquely predict second language learning rate and willingness to communicate in adults. *Journal of Cognitive Neuroscience, 31*(1), 78-94.

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1-24.

Quille, K., & Bergin, S. (2018, July). Programming: predicting student success early in CS1. a re-validation and replication study. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (pp. 15-20).

Rayner, K., Schotter, E. R., Masson, M. E., Potter, M. C., & Treiman, R. (2016). So much to read, so little time: How do we read, and can speed reading help?. *Psychological Science in the Public Interest, 17*(1), 4-34.

Rist, R.S. (1995). Program structure and design. *Cognitive Science, 19*, 507–562.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137-172.

Roschelle, J., & Greeno, J. G. (1987). *Mental models in expert physics reasoning.* CALIFORNIA UNIV BERKELEY.

Rouse, W. B., & Morris, N. M. (1986). On looking into the black box: Prospects and limits in the search for mental models. *Psychological Bulletin, 100*(3), 349.

Sauter, V. L. (1986). Predicting computer programming skill. *Computers & Education, 10*(2), 299-302.

Scarlett, R. (2023, March 2). *Why Python keeps growing, explained*. GitHub. Retrieved from https://github.blog/2023-03-02-why-python-keeps-growing-explained/

Scholtz, J., & Wiedenbeck, S. (1990). Learning second and subsequent programming languages: A problem of transfer. *International Journal of Human-Computer Interaction, 2*(1), 51-72.

Seow, P., Looi, C. K., How, M. L., Wadhwa, B., & Wu, L. K. (2019). Educational policy and implementation of computational thinking and programming: Case study of Singapore. *Computational Thinking Education*, 345-361.

Shevy, M. (2008). Music genre as cognitive schema: Extramusical associations with country and hip-hop music. *Psychology of Music, 36*(4), 477-498.

Shute, V. J. (1991). Who is likely to acquire programming skills?. *Journal of Educational Computing Research, 7*(1), 1-24.

Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., ... & Brechmann, A. (2014, May). Understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering* (pp. 378-389).

Simon, S. Assignment and sequence: why some students can't recognise a simple swap. In

Proceedings of the 11th Koli Calling International Conference on Computing Education Research, Koli Calling (Vol. 11, pp. 10-15).

Simon, S., Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., ... & Tutty, J. (2006). Predictors of success in a first programming course. In *Conferences in Research and Practice in Information Technology* (Vol. 52, pp. 189-196). Australian Computer Society.

Sirkiä, T. (2012). *Recognizing programming misconceptions-an analysis of the data collected from the uuhistle program simulation tool* (Master's thesis).

Sirkiä, T., & Sorva, J. (2012, November). Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research* (pp. 19-28).

Sleeman, D., Putnam, R. T., Baxter, J., & Kuspa, L. (1986). Pascal and high school students: A study of errors. *Journal of Educational Computing Research, 2*(1), 5-23.

Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct?. *Communications of the ACM, 29*(7), 624-632. http://dx.doi.org/10.1145/6138.6145

Staggers, N., & Norcio, A. F. (1993). Mental models: concepts for human-computer interaction research. *International Journal of Man-Machine Studies*, *38*(4), 587-605.

Steedman, M. (1996). The blues and the abstract truth: Music and mental models. *Mental models in Cognitive Science*, 305-318.

Swaab, T. Y., Camblin, C. C., & Gordon, P. C. (2004). Electrophysiological evidence for reversed lexical repetition effects in language processing. *Journal of Cognitive Neuroscience, 16*(5), 715-726.

Swaab, T. Y., Ledoux, K., Camblin, C. C., & Boudewyn, M. A. (2012). Language-related ERP components.

Tanner, D., Inoue, K., & Osterhout, L. (2014). Brain-based individual differences in online L2 grammatical comprehension. *Bilingualism: Language and Cognition, 17*(2), 277-293.

Tanner, D., McLaughlin, J., Herschensohn, J., & Osterhout, L. (2013). Individual differences reveal stages of L2 grammatical acquisition: ERP evidence. *Bilingualism: Language and Cognition, 16*(2), 367-382.

von Mayrhauser, A., & Vans, A. M. (1994). *Program Understanding: A Survey*. Colorado State Univ.

Vu, K. P. L., Hanley, G. L., Strybel, T. Z., & Proctor, R. W. (2000). Metacognitive processes in human-computer interaction: Self-assessments of knowledge as predictors of computer expertise. *International Journal of Human-Computer Interaction, 12*(1), 43-71.

Watson, C., & Li, F. W. (2014, June). Failure rates in introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (pp. 39-44).

Whipkey, K. L. (1984). Identifying predictors of programming skill. *ACM SIGCSE Bulletin, 16*(4), 36-42.

Wills, J. A. (1982). Are programming and natural language skills related. *Communications of the ACM, 25*(3), 221.

Wilson, J. R., & Rutherford, A. (1989). Mental models: Theory and application in human factors. *Human Factors, 31*(6), 617-634.

Wittrock, M. C. (1974). Learning as a generative process. *Educational Psychologist, 11*(2), 87-95.

Wu, Q., & Anderson, J. R. (1990). *Problem-solving transfer among programming languages.* CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND

PSYCHOLOGY PROJECT.

Zatorre, R. J., & Krumhansl, C. L. (2002). Mental models and musical minds. *Science, 298*(5601), 2138-2139.

Zwaan, R. A. (2016). Situation models, mental simulations, and abstract concepts in discourse comprehension. *Psychonomic Bulletin & Review, 23*, 1028-1034.

Zwaan, R. A., & Radvansky, G. A. (1998). Situation models in language comprehension and memory. *Psychological Bulletin, 123*(2), 162.