TECHNISCHE UNIVERSITÄT
CHEMNITZ

The effects of shortcuts in program comprehension

Master Thesis

Submitted in Fulfilment of the
Requirements for the Academic Degree
M.Sc.

Dept. of Computer Science
Chair of Software Engineering

Submitted by: Vipul Oswal
Matrikel Nr:
Date: 10.09.

Supervision:
Prof. Dr. Janet Siegmund
Ms. Arooba Aqeel

Abstract

Background: There has been a lot of research on how to teach and learn different programming idioms, but not much on the effects of shortcuts in program comprehension.
Objective: We must first analyze how students interpret statements with shortcut codes in order to develop more effective teaching methods for them. Investigating this phenomenon can help us pinpoint the elements that may influence students' comprehension of shortcuts.
Method: We conducted an eye-tracking study to monitor students' visual attention while they participated in understanding simple shortcut programs. The study included 21 students in all, and behavioral and visual-attention data were gathered.
Results: We discovered clear evidence that was knowing shortcuts influences how well programs are comprehended. Regarding visual attention, we found that after being taught the shortcut, they were able to comprehend the program with ease.
Conclusion: When students are introduced to shortcuts, there is a difference in how quickly they understand algorithms. The same group's visual attention is equally indistinguishable for short-term understanding.
Future Work: Furthermore, research is required to draw results that are more general, for instance using more complex shortcut code or allowing students to write their own code using shortcuts. A more accurate understanding of the core cognitive process underlying program comprehension may be gained by measuring the cognitive load, such as pupil dilation.

Acknowledgment

I want to thank my supervisor Arooba Aqeel and professor Prof. Dr. Janet Siegmund for giving me this chance as well as for their support and advice when I was conducting research for my thesis. Additionally, I would like to thank Chemnitz University of Technology for providing the resources. I would not have been able to effectively finish this task without their wise guidance and supportive assistance. My sincere gratitude to these people for their contributions. In conclusion, I would like to thank everyone who contributed to the study and provided thoughtful comments.

Contents

List of Figures

List of Tables

List of Abbreviations

API Application Programming Interface

GPU Graphics Processing Unit

CPU Central Processing Unit

GPGPU General Purpose Graphics Processing Unit

RTX Ray Tracing Texel eXtreme

CUDA Compute Unified Device Architecture

LOC Line Of Code

BCS Basic Control Structure

HPC High Performance Computing

AI Artificial Intelligence

PTX Parallel Thread Execution

ISA Industry standard architecture

NVRTC Runtime Compilation Library for CUDA

JIT Just In Time

OS Operating System

OSX Operating System Extended

LLVM Low Level Virtual Machine

HCI Human Computer Interaction

EEG Electroencephalogram

PCCR Pupil Centre Corneal Reflection

EOG Electrooculography

ML Machine Learning

SVM Support Vector Machine

KNN K-Nearest Neighbors

Hz Hertz

AOI Area Of Interest

TTFF Time to First Fixation

CFS Cognitive Functional Size

RQ1 Research Question 1

RQ2 Research Question 2

ODT Outlier Detection Techniques

PPMC Pearson Product Moment Correlation

1 Introduction

1.1 Motivation and Relevance

This thesis aims to share information about students' comprehension of shortcut code. We investigated how the shortcuts affected students' behavior and visual attention to complete a specific task. We also trained a group of students and divided them into two groups to gain additional insights. If we discuss why understanding some unusual line of code is valuable, we can notice that a programmer encounters multiple code fragments while learning or developing algorithms. This motivates us to explore the possibilities of the problems that occurs when a program encounters unusual statement. At that moment, the awareness of the shortcuts and syntax is challenged by the large code base. This is crucial for programmers since they frequently encounter different ways to carry out various operations. When writing code for comparable operations, various programmers use different writing styles and methodologies because a variety of solutions may be able to yield the same outcome. However, in the end, what matters is whether other programmers can understand what has been accomplished after utilizing that specific statement. This encourages us to investigate whether there are any appreciable differences in how well various experienced groups understand any particular syntax or shortcuts embedded in different aspects of code. It also gives us information about their programming knowledge and problem-solving abilities. On the other side, if a group of students is taught the shortcuts, can they understand the code and how effectively they could use those shortcuts in various contexts to get results? Often programmers with a grasp over any particular language will produce the result by using some shortcuts or few logical statements as compared to the novice or intermediate experience level programmer. Their statement can be more efficient with optimal time and space complexity. Even when the programmer has encountered some shortcuts in their past experience, are they still able to comprehend them?

1.2 Research Objective

In this paper, our objective of work is to present some insights into whether students are able to comprehend shortcut code. To accomplish our task, we examined how the shortcuts affected students' behavior and visual attention. Further, to gain more insights, we included trained a group of students and then categorized them into two separate groups.
To add clarification to our objective, we address the research questions as.

1. While comprehending the source code do program shortcuts affect students' response time and correctness?
2. While comprehending the source code program shortcuts affect students' visual attention?

We will touch the following hypothesis and later conclude whether training the student add much more value in understand the code with shortcut statement(s)?
Behavioral Data:
H_0: There is no significant difference in "Response Time" between both groups.
H_1: There is significant difference in "Response Time" between both groups.
H_2: There is no significant difference in "Correctness" between both groups.
H_3: There is significant difference in "Correctness" between both groups.
Visual Data:
H_4: There is no significant difference in "FirstPassDuration" between both groups.
H_5: There is significant difference in "FirstPassDuration" between both groups.
H_6: There is no significant difference in "SecondPassDuration" between both groups.
H_7: There is a significant difference in "SecondPassDuration" between both groups.
H_8: There is no significant difference in "noOfFixtionsInAOI" between both groups.
H_9: There is significant difference in "noOfFixtionsInAOI" between both groups.

1.3 Research Design

Each participant was familiar with the basics of Python. The snippets will be shown sequentially to each participant, and behavioral and visual data were gathered that will be essential to the analysis and conclusions. There were 21 participants altogether, and 11 of them took the pretest to become familiar with the shortcuts. In the study, we tracked the user's eye movement using a Tobii Pro Fusion eye tracking device. The study was carried out on PyschoPy and at the end of each exercise students were then asked to choose the right response from the four options after each exercise in the research. The behavioral and correctness data were generated at the completion of each test.

1.4 Research Implications

The main goal is to find out whether there is any significant difference between understanding shortcuts (special syntax) used in any code amongst different experienced groups. It also provides us insights into their knowledge of the programming language and problem-solving skills. On the other hand, if the shortcuts are taught to a group of students, can they comprehend the code and how well they could perform using those shortcuts in different scenarios to achieve different results?
The following sections will provide details of our research and findings.

2 Literature Review

2.1 Program Comprehension

2.1.1 Reading Order of Programmers

A key task in software development, programmers' comprehension of source code has been the subject of extensive research over the past few decades [96, 97]. Program comprehension, the fundamental cognitive process, is a requirement for all ensuing programmer tasks, including testing, debugging, and maintenance. There are two basic approaches that programmers use to understand software, according to previous studies. When programmers lack the subject expertise, experience, or context necessary to effectively grasp source code, bottom-up comprehension is used [63]. Instead, in order to get a comprehensive understanding, users must comprehend individual source code lines and statements and integrate their semantic meaning (i.e., chunking [80]). When programmers utilize prior knowledge or domain expertise for a productive hypothesis driven comprehension process [86], for as when using variable Identifier [14, 61].
Even while these existing comprehension models are supported by some evidence, there are still some unknowns, such as when and how programmers can use top-down comprehension. Because it involves internal cognitive processes, measuring program understanding is inherently challenging [82]. Conventional approaches, such think-aloud protocols or task efficiency testing, are unable to offer profound insights into the cognitive mechanisms underpinning program comprehension [61].
Observing how programmers read source code is a crucial part of understanding programs. Eye tracking has been effective for observing programmers reading source code and addressing such basic research issues as program comprehension (e.g., [19, 79, 99]). For instance, Sharif and Maletic used eye tracking to replicate a traditional study and discovered that name style influences program comprehension in that programmers can comprehend different styles [13, 79, 61].
The linearity of the reading order may be a measure of how well programmers understand source code, according to earlier studies [19]. Multiple eye-gaze measurements to evaluate the linearity of reading order were described in the pioneering work by Busjahn et al. They demonstrated that both skilled programmers and beginner programmers scan source code less linearly than natural text [19]. According to this study, understanding source code is a skill that must be acquired via practice [61].
In this study, they delve deeper into the significance of reading order linearity for program comprehension. They specifically want to know how programmers' read-

ing habits are influenced by their comprehension strategies and the linearity of the source code itself. To more precisely assess program understanding with eye tracking, it is essential to comprehend all the elements that affect programmers' linearity of reading order. To do this, they used novice and experienced programmers in a non-exact replication of the Busjahn et al. and Peachock et al. experiments [61].
To determine the impact of source code linearity, programmer experience, and comprehension technique on the reading order of source code. The research was motivated by Busjahn et alpioneering's study [19] and the replication study by Peachock et al [60]. According to their findings, expertise and comprehension technique appear to have less of an impact on programmers' reading order than the linearity of source code. They appear to have discovered a turning point when programmers shift from a linear reading order to a reading order following the execution order, with their intermediate programmers' experience level falling between the two prior studies. The substantial influence of linearity implies that the source code's organization should meet the expectations of the programmer in order to prevent needless eye movements and maybe improve program comprehension. They plan to go more deeply into the magnitude of the effect of source code linearity, programmer expectation, and experience level in further investigations [61].

2.1.2 Abstraction and Complexity in Program Comprehension

The talent of programming has several facets. It is generally acknowledged that programming involves more than just writing code and also includes important abilities like understanding programs, designing, and testing [33, 49, 50, 93, 94, 103]. The component of program comprehension that this article focuses on is one that is stressed in many pedagogical approaches to teaching beginners about programming [38, 45, 57, 77, 105], but it is equally important for specialists like seasoned software developers [42, 89]. Previous research [15, 18, 47, 62, 75] examined several facets of program comprehension, including how students progress from textual pieces to higher levels of abstraction and how program parts express overarching objectives. The idea that comprehension can be characterized as the creation of a mental image of the program is shared by everyone [76].
Research on human cognition, particularly Chunking [20, 55] and Schema Theory [73], which explains how students deal with a large number of concepts and integrate them to form a mental model of the dynamic elements of program execution, has a significant influence on research on program comprehension. The idea of plan-schemata, which was established in early Computing Education Research (CER) research, is essential to comprehension (e.g., References [67]). Planschemata function as "a library of archetypal answers to issues as well as mechanisms for coordinating and assembling them" [85]. The following terminology is used in this article:

1. The idea that comprehension can be characterized as the creation of a mental image of the program is shared by everyone [76].

2. (Standard) Plans relate to code portions since they are programming language

realizations of a plan-schema. It depends on the context and instruction what constitutes a standard plan. In this work, they take guard, average, counter, and sentinel plans—extracted from Soloway and Ehrlich [87]—as examples of standard plans.

3. According to Soloway's definition, customized plans are those that have been altered, such as applying a FindMaximum-plan on an array while storing the index of each candidate [85].

4. Code parts that do not match any plan-schemata are referred to as unplan-like code.

A plan-composition strategy is one of the techniques students adopt, according to the literature on how they write code sections [85]. Students might order (adjacent) plans, for instance, where one plan's execution is independent of the others or where one plan's output is used as an input for subsequent plans. Another tactic is to combine all of the plans into a single code section, where each plan's execution is sandwiched between the others. A FindMaximum-plan and a FindMinimum-plan having the same loop code would be an example. Students who employ merged plan-compositions have been shown to create code that is more prone to errors [26, 87], yet they still believe it to be "better" [27]. It has been proposed [23] that due to decreased element interactivity [92], i.e., plans that require learners to absorb them simultaneously, sequenced programs may be easier to understand than merged ones. Therefore, it makes sense that sequenced composition is advantageous for comprehension as well as for producing code [27]. They are curious to know if plan-schemata and the structure of code sections have any direct impact on how well code is understood. Plans in programming have been shown to be more effective than unfamiliar code in the past [87], but no direct research has been done on the impact of plans, tailored plans, unplan-like code, or the structure of such code sections on learners' comprehension behavior. To close this gap, they carried out a study with a purposeful sample of people who had been carefully chosen to have access to plan-schemata. This article offers three contributions: First, we offer insights into how plan-schemata may impact the process of program comprehension. Second, they look into the impact of such solutions' composition on program understanding. They accomplish this by utilizing cutting-edge research techniques and eye-tracking data. They adopt plans as the unit of measurement, focusing on more significant pieces of information rather than saccades and transitions between keywords or syntactic parts. This methodology makes it easier to do the study because it involves less precise eye-tracking techniques and yields more insightful data on the program comprehension process. The setting for a study design is as a result our third contribution: They can utilize a mix of non-invasive sampling of high-quality eye-tracking data enhanced with insights from retrospective interviews [31] by using the model-building assumption (see Section 3.4) and its verification.
Summary. Regarding RQ 1: How does program understanding differ depending on whether plan-schemata are available? Plans were discovered to be more rapidly

grasped than personalized plans. Therefore, they assume that plan-schemata made chunking happen more quickly, which sped up the understanding process. Qualitative interview data shows that students thought customised plans were comparable to plans, which, in contrast to code parts that don't resemble plans, may have helped them understand programs. A statistically significant effect, however, could not be seen in this case. Below, some explanations for the ambiguous findings are presented.

Regarding RQ 2: How does program comprehension depend on the structure of code sections? When compared to sequenced code, they found that code given in a merged form greatly increases the frequency of transitions between various code blocks. Data from interviews confirmed that this effect was caused by a greater challenge in finding interconnections between code blocks in merged compositions: Element interactivity is increased through combined code composition. According to the fact that these effects were observed for both code blocks with the same level of familiarity as well as for code blocks with different levels, they are independent of familiarity. However, they didn't see any strong evidence to suggest that this also impacts how long it takes to understand a code.

Although the study's materials were piloted (see Section 3.3), several unintended consequences nonetheless had an impact on the results. They advise considering certain factors when using the setup we provided: In Omega, they noticed that some participants found a particular idea that wasn't in the companion program to be surprisingly challenging. They also found that for programs that were either too simple or too tough, there was no discernible influence of code composition on the amount of time required to perform the initial input-to-output conversion operation. It is important to match students' cognitive resources with the task difficulty if one wants to research the impact of a prospective feature that makes it more difficult. It is important to challenge students without overwhelming them [74].

They found that understanding plan-schemata had a good overall impact on program comprehension. This study thus emphasizes the significance of encouraging the growth of such plan-schemata in beginner programmers once more. They also saw a code composition effect and discovered signs that a combined code composition method may make understanding more difficult by making elements more interactive because different code parts must be analyzed together rather than individually. However, more research is necessary.

This work makes a methodological addition by presenting a setup that enables the use of eye-tracking data to analyze program understanding. Multiple time-sensitive input-to-output conversion tasks allowed them to see evidence that participants did create a mental model during the first task, which they then used in subsequent tasks. They recommend that this assumption can be applied in future efforts to deliver high-quality data in program comprehension studies using the approach we gave.

2.1.3 Program Comprehension in professional developers

Understanding programs is a crucial task in software maintenance. According to Fjeldstadand Hamlen [28], it takes up around half of the time that developers spend on maintenance. Program comprehension, according to Singer et al. [84], primarily occurs prior to modifying code because developers must examine source code and other artifacts to locate and comprehend the portion of the code that is pertinent to the desired change. Developers may use different approaches to comprehend software depending on their personalities, experiences, skills, and the work at hand. This study aims to examine the current state of program comprehension practice and discover how programmers in industry interpret programs. Our goals include deepening our understanding of program comprehension practice, investigating the application of research findings in practice, confirming the outcomes of related studies, and addressing the shortcomings of past research. Additionally, the hypotheses derived from our observations serve as the basis for a research agenda that is motivated by unmet needs [70].
There are gaps in earlier empirical investigations of program comprehension techniques, necessitating a more thorough, current investigation. For instance, the investigations by Singer et al. [84] and Fjeldstad and Hamlen [28] are relatively ancient. New programming languages, like Java, and methodologies, like agile development and opensource development, have gained popularity since their study. Developers from a single business are studied by LaToza et al. [46] and Robillard et al. [68] conducted a lab investigation with only five developers. Their study, which has a bigger sample size of 28 developers, examines developers who work in various firms of various sizes and across a variety of technologies. Additionally, they use a distinct approach that enables thorough explanations of the reasoning and ideas behind observed behavior [70].
Their study, which has a bigger sample size of 28 developers, examines developers who work in various firms of various sizes and across a variety of technologies. Additionally, they use a distinct approach that enables thorough explanations of the reasoning and ideas behind observed behavior [70].
This study aims to investigate qualitatively how program understanding is carried out in the software business. This entails researching the practical use of program comprehension tools, developing theories regarding industrial program understanding, and putting them to the test. Since the distribution and features of the entire population are unknown and their sample is, statistically speaking, somewhat tiny, they refrain from quantifying several aspects of comprehension. Furthermore, they don't specifically pursue novel theories. They examine three key areas in order to organize the research and concentrate their efforts: the tactics developers use, the information they interact with or that is missing, and the tools they employ. The question that they touched were [70]

1. RQ1: Which strategies (including steps and activities)do developers follow to comprehend programs?

2. RQ2: Which sources of information do developers use during program comprehension?

3. RQ3 : Which information is missing?

4. RQ4: Which tools do developers use when understand-ing programs and how?

One of their most intriguing results is that developers often try to put themselves in the shoes of end users. They saw developers examine the user interface behavior and contrast it with the desired behavior. This method seeks to comprehend program behavior and obtain initial cues for deeper program investigation. It offers an alternative to troubleshooting and reading source code. Additionally, programmers occasionally strive to avoid comprehending their own code. Instead, they copy the source code and modify it to complete the work at hand. Cloning avoids understanding the potential effects of directly altering code. Due to the time and mental work required, developers appear to choose solutions that prevent comprehension wherever possible. Program understanding is not viewed as a goal in and of itself, but rather as a necessary step to complete certain maintenance responsibilities. They discovered that standards and expertise are crucial facilitators for quickly becoming comfortable with an unfamiliar program and identifying entry points for further research when software architecture and code need to be investigated. Depending on their work situation, the majority of observed developers pick from a variety of structured comprehension tactics (such as adhering to a problem-solution-test work pattern). The type of work at hand, the kind of program to understand, prior understanding of the program, and the developer's overall experience make up context [70].

2.2 What (Else) Should CS Educators Know?

The qualifications needed to teach in a scientific discipline differ from those needed to practice or do research in it. While the work of the former requires extensive knowledge and skills in the field itself, that of the latter also requires the ability to accurately and reliably communicate this knowledge to others, to teach the aforementioned skills, to provide perspective, and to arouse the interest, curiosity, and enthusiasm of the students. All of this necessitates the educator becoming more of a scientific intellectual, at least in terms of the relevant discipline. While some of this is a matter of personality and innate aptitude, they contend that some of it may be learned by being exposed to material that extends beyond the technical core components of the area [30].
As these problems emerge in the CS sector, they examine them in further detail. They specifically point out some of the additional content that CS educators should learn in addition to what is typically covered in an undergraduate CS curriculum. They have created an undergraduate course out of this content as an intriguing byproduct of their work, and they have some experience teaching it [30].

They have suggested topics for a course for CS educators in the more practical section of this post, even mentioning that some of this should be interesting to people outside the CS community. But they haven't been clear enough on how such a course should be taught, how the hours should be split up, and so on. This is done intentionally because they don't think there has been enough experience in these areas to offer definitive, strict suggestions. In any event, we think instructors should design the course as they see fit [30].
They have some experience putting the concepts mentioned here into practice. Such courses have been taught by Gal-Ezer at The Open University and Tel-Aviv University, both in Israel. The course uses a reader with about 30 papers and a custom created study guide. They find it quite helpful to have students write extensive term papers, and they also want to have a number of invited lecturers in the hopes that the students will benefit from both what they have to say and how they express it [30].
One of the key takeaways they took away from teaching the course was the necessity of students having a solid background in computer science. This point cannot be emphasized strongly enough. One student in the class was an electrical engineer, another only had a passing link to computing because she used computers in general education, and a third student had CS knowledge that was 25 years old. Simply put, these students did not fit in. In conclusion, even though our suggestions are not entirely full, they hope that their work will aid in the creation of such courses and in choosing their structure and contents [30].

2.3 Shortcuts

Through the use of written-down sounds and symbols, language allows individuals to communicate. People learn language as a result of their life experiences, but linguistics, the scientific study of languages, goes far further in its examination of word patterns and implications. Computer programming languages, the subject of this text, fall under this academic discipline. Programming languages can be thought of as artificial languages created by men and women initially for the purpose of communicating with computers, but also, and perhaps more importantly, for the purpose of communicating mathematics among people, as opposed to natural languages, with which we communicate our thoughts and feelings. Programming languages make use of many linguistic concepts and terminology. For instance, syntax and semantics constitute the majority of language definitions [37].

2.3.1 Syntax

The syntax of a programming language is used to express a program's structure, but the content of the program itself is ignored. It involves applying a set of guidelines to check the order in which symbols and instructions are utilized in a program. The main function of Grammars, which are rewriting rules, is to create programs. There

are just a few number of pre-formed grammatical categories, words, and rules that make up grammar. These formal and informal techniques can be used to comprehend a programming language's syntax [37]:
Lexical syntax These syntax describe the rules for basic symbols like as identifiers, literals, punctuation, and operators.
Concrete syntax This syntax describes the lexical units, usually referred to as tokens, of a computer language. It emphasizes the expression's appearance further.
Abstract syntax This syntax only conveys the application's most crucial information.

2.3.2 Syntax Error

In the realm of computer science, a syntax error is a mistake made by a programmer in the syntax of a coding or programming language. Finding syntax problems in a program is the job of a piece of software called a compiler. Before the software is produced and run, the errors must be rectified by the programmer. Programming languages are intended to be extremely exact and unambiguous, therefore if they disregard or violate the language's terminology, they commit a syntax error. As a result, the software will be unable to run and will instead generate a helpful error message [35].

2.3.3 Semantics

The meaning of syntactically sound statements is explained by the semantics of a language. When it comes to natural languages, this involves linking specific words and phrases to specific concepts, thoughts, and feelings. The study of semantics focuses on how computers respond to commands in programming languages. This behavior might be demonstrated, for instance, by detailing how a program would operate on a real or imagined computer or by showcasing the relationship between the code's input and output. The term "semantics" is quite helpful when attempting to comprehend how a programming language's syntax and computation model relate to one another. It emphasizes the interpretation of a program so that the programmer can predict or understand its results with ease [35]. Utilizing a functional mapping of syntactic constructions to the computational model, syntax-directed semantics is used. Some approaches to describing the semantics of a computer language include algebraic semantics, axiomatic semantics, operational semantics, and denotation semantics [29, 98].
Algebraic semantics Information and language structures are expressed using algebraic notation in algebraic semantics. Providing a formal, axiomatic explanation of the attributes of various sorts of objects and actions on those objects is the aim of the algebraic approach to semantics.
Operational semantic Operational semantics aims to convey the meaning of a program starting from a specific state by examining its end result, or the state in which the memory remains after the program has been executed. This can be

accomplished by checking the memory's condition once the program has finished running.
Denotational Semantics This semantics is predicated on the idea that a program may be viewed in a manner similar to that of a mathematical function, i.e., that a program's impact may be perceived as a mathematical function in state.
Axiomatic Semantics Axiomatic Semantics examines if a given program is partially correct (with regard to pre and post-condition). It creates assertions about an association to be checked at each stage of the program's execution in order to determine the application's intended use (i.e., implicitly).

2.3.4 Semantic Error

Even if your software is successfully compiled, it could still be challenging to get it to produce the desired outcomes. A semantic error is when a statement has the intended meaning but does not actually have it. This can happen even when the syntax of the statement is valid. A semantic error, as opposed to a syntactic error, is a misunderstanding of the intended meaning. This kind of issue will allow a program to run, but the outcome it generates won't be accurate.

Syntax Analysis	Semantic Analysis
Process of analyzing a string of symbols either in natural languages or data structures conforming to the rules of formal grammar	Process of checking whether the generated parse tree is according to the rules of the programming language.
Parser performs syntax analysis	Semantic analyzer performs semantic analysis
Second phase of the compilation process	Third phase of the compilation process
Takes the tokens as input and generates a parse tree as output	Checks whether the parse tree is according to the rules of the language
Generates a parse tree	Generates an annotated syntax tree

Table 2.1: Difference between Syntax and Semantics

3 Methodology

3.1 Study Pattern

Choosing whether to conduct the quantitative usability research between or within individuals is a component of the experimental design [90]. There are two methods that could be used:

Between-subjects study experiment: This study's design entails giving each test subject a unique user interface. In this manner, a single user interface is used by each test subject [90].

Within-subjects study experiment: In this study design, all of the user interfaces you're assessing are presented to each test subject. Each test subject will so experience every circumstance [90].

Usually, independent variables and dependent variables are used in quantitative usability studies. Dependent variables are those that are measured in relation to changing independent factors, whereas independent variables are those that researchers manipulate. The code snippets in our example would be independent variables, and the behavioral and visual data that the user produced while the accomplishment of the task would be the dependent variables. The aim of the usability research in this scenario may be to ascertain whether the dependent variables change or stay the same while the independent factors are constant. As our independent variable is same for all the participants we follow the within subjects study experiment.

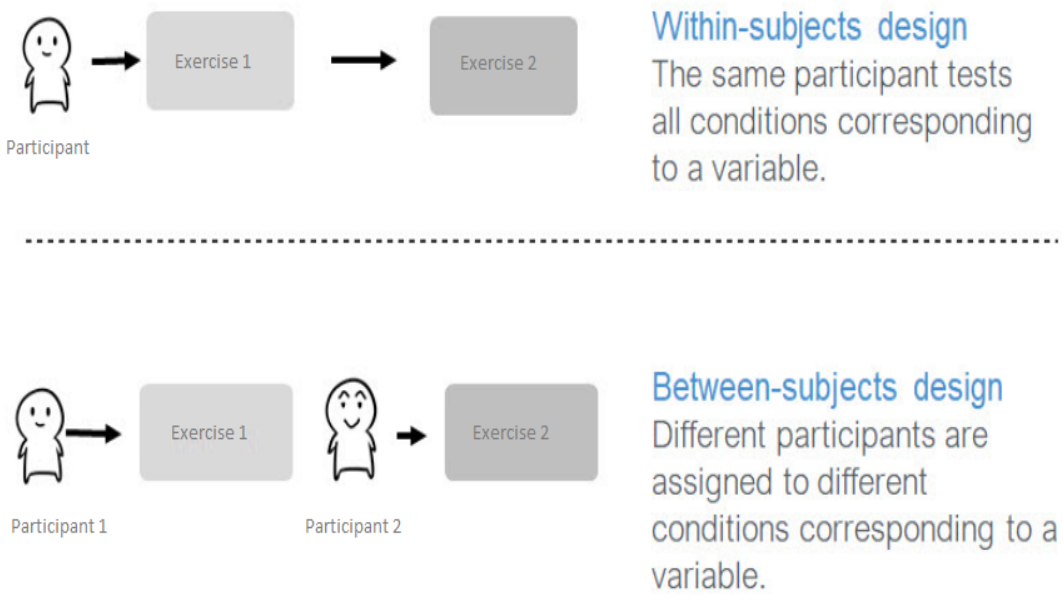

Figure 3.1: Difference between With-In and Between Study Design [17]

3.2 Selection of Code Snippets

The idea of a code's Basic Control Structure (BCS) and cognitive weights are important factors when selecting code fragments for study. One of the main causes of this is that it is challenging to create an experiment that can gauge the mental effort required to comprehend the impact of different programming constructs and how they interact. Since there are countless variations, we are unable to choose and compare any random code fragments of different programming structures in such tests. With the exception of changes that are inherent in syntax, we demonstrated with examples the many factors and challenges involved in choosing code snippets for various programming constructions [39].
A number of psychological code snippets approach can be carried out to obtain meaningful code fragments. A series of necessary measures not just to verify the cognitive weights of various programming constructs, but there to develop reliable metrics for code complexity. These will be helpful for understanding the cognitive load needed to acquire the principles of shortcuts used in our experiments [39]. To assess the complexity concealed within the software, numerous metrics like Line of Code (LOC), Halstead measures, and McCabe's cyclometric measurements have been proposed in the past [53]. All of these metrics only account for certain facets of complexity while neglecting others [39].
None of these, however, adequately reflects the mental effort needed to comprehend the software code, which is a human aspect of software complexity. To reflect the

complexity of the software, Shao and Wang introduced a new measure of cognitive weights size in 2003 and the cognitive weights of 10 Basic Control Structure (BCS) [78]. Following a series of student-subject psychological investigations, Wang adjusted the cognitive weight of the 10 Basic Control Structure (BCS) in 2006 [102], [101]. After that, other metrics based on the cognitive weights introduced in 2003 and 2006 [24], [40], [56] were proposed by various researchers. In 2007 Gruhn and Laue identified a number of problems with Wang's psychological experiments [34]. He also discussed the difficulties assessing the cognitive weights of recursion Basic Control Structure (BCS) because of its special characteristics. Additionally, he recommended that we add three additional control structures, namely locks, exceptions, and internal exits, to the table set of Basic Control Structure (BCS). He listed a few safety measures that should be considered while planning any such trials. The same type of study was carried out by David Admino in 2015, and he came to a quite different conclusion as Wang [9] in an unpublished paper (but one that is accessible on the Research Gate website). In a paper published in 2017, Ajmi et al. conducted an experiment and demonstrated that the complexity of a portion of code depends on a variety of factors, including the predicate's expression style and the idioms used (for example, in looping structures) [10]. The latter aspects were not taken into account in the software code complexity computation in the original Wang and Shao metrics in 2003 and 2006 [78], [102].

3.2.1 Basic Control Structure (BCS) Cognitive Weights

The idea of cognitive functional complexity of software was first suggested in 2003 by Yingxu Wang [78]. Cognitive weights are assigned to the BCS basic control structures in this metric. Basic Control Structure (BCS) is a collection of fundamental and necessary flow control techniques that are utilized in the construction of software's logical architecture [39].
In these measures, the weights of a Basic Control Structure (BCS) are multiplied if they are embedded in another Basic Control Structure (BCS) or added together if they are in series to determine the component's overall cognitive weight. The sum of the cognitive weights of each of a software component's q linear blocks that make up an individual Basic Control Structure (BCS) is known as the component's total cognitive weight or Wc. Given that each block may have m layers of nesting Basic Control Structures (BCSs) and n linear Basic Control Structures (BCSs) in each layer, the equation can be used to determine the overall cognitive weight, Wc (1) [39].

$$Wc = \sum_{j=1}^{q}[\prod_{k=1}^{m}\sum_{I=1}^{n}(Wc(j,k,i))] \dots (1)$$

The weights for the various Basic Control Structure (BCS) are assigned in this metric, as stated in table 3.1. These weights are dependent on how difficult it is for a person to understand these Basic Control Structure (BCS). A fundamental software component's Cognitive Functional Size (CFS), which is determined by its

single method and the sum of its inputs and outputs $N_{i/10}$, is determined by the overall cognitive weight, i.e.: [39].

$$Sf = N_{i/10} * Wc \ldots (2)$$

Category	Basic Control Structure (BCS)	Cognitive Weights (Wc)
Sequence	Sequence	1
Branch	If then else	2
	Case	3
Iteration	For-loop	3
	Repeat-loop	3
	While-loop	3
Embedded Component	Function call	2
	Recursion	3
Concurrency	Parallel	4
	Interrupt	4

Table 3.1: Cognitive Weights of different Basic Control Structure BCS-Wang 2003 [78]

However, Wang proposed revised weights for several Basic Control Structure (BCS) in 2006, which are listed in Table 3.2 [102]. Wang altered the Basic Control Structure (BCS) weights, but the technique for determining the software's overall cognitive complexity has not changed. In a previously unpublished article (but one that is accessible on the Research Gate website), David Admino ran similar experiments in 2015 and came to a fairly distinctive conclusion than Wang [9], which can be seen in table 3.3 [9].
The guidelines listed below were provided by Gurhn and Laue for carrying out studies of this nature [34]. These consist of [39].

- Code length, variables-Number, types, and names are constant.
- Multi-language experimentation.
- Various responses with varying levels of experience.
- Adequate repetitions.
- Adhering to the rigid experimental investigation methodology.

Category	Basic Control Structure (BCS)	Cognitive Weights (Wc)
Sequence	Sequence	1
Branch	If then else	3
	Case	4
Iteration	For-loop	7
	Repeat-loop	7
	While-loop	8
Embedded Component	Function call	7
	Recursion	11
Concurrency	Parallel	15
	Interrupt	22

Table 3.2: Cognitive Weights of different Basic Control Structure BCS- Wang 2006 [102]

Category	Basic Control Structure (BCS)	Cognitive Weights (Wc)
Sequence	Sequence	1
Branch	If then else	2
	Case	2
Iteration	For-loop	11
	Repeat-loop	10
	While-loop	6
Embedded Component	Function call	Not Calculated
	Recursion	7
Concurrency	Parallel	Not Calculated
	Interrupt	Not Calculated

Table 3.3: Cognitive Weights of different Basic Control Structure BCS-David Admino 2015 [9]

3.2.2 Line of Code (LOC)

We think that rigorous adherence to the premise of the same code length for all snippets to be compared may not be the wisest course of action. First of all, dif-

ferent languages may display varying lines of code (LOC) for the identical logic expression. Second, Line of Code (LOC) differences are present because of the fundamental structure of different Basic Control Structure (BCS). For instance, the Line of Code (LOC) of later in comparable code snippets of sequential and if-else will differ because of the presence of at least one condition statement and potentially two pairs of curly braces in if-else, which increases the Line of Code (LOC) of later [39]. When using nested if-else instead of a single logical expression in condition statements, Ajami et al have demonstrated that the same logic solves more quickly than when using a single logical expression, despite the fact that nested if-else has a far higher Line of Code (LOC) [10]. We are not advocating the complete abolition of this code length factor. While we continue to maintain that the Line of Code (LOC) of code snippets shouldn't fluctuate significantly, all we are saying is that these rules cannot be legally enforceable for carrying out such trials [39].
It adds the additional requirement of having the same quantity and variety of operators and variables. By doing this, we eliminate the variation that may result from changing Line of Code (LOC), using various operators, and using varied variable types. Because we think, based on experience, that not all operators are the same and that they do differ in complexity [39]. The Basic Control Structure (BCS) Cognitive Weights and Line of Code (LOC) in our study is declared in table 4.1

3.3 CUDA framework GPU

Python is the core for data analytics and deep learning applications. NVIDIA, on the other hand, showed a great commitment towards making a contribution to the python ecosystem to enhance the parallel performance of GPU to improve the compatibility with standardized libraries, tools, and applications. Python CUDA ecosystem offers full access to the CUDA host API. It makes it easier for developers to use NVIDIA GPUs [58].

What is CUDA Python?
CUDA from NVIDIA Python offers Cython bindings and Python wrappers for the driver and runtime API for pre-existing toolkits and packages to make GPU-based accelerated processing simpler. One of the most widely used programming languages is Python, which is used for applications in data analytics, deep learning, engineering, and science. In order to provide complete coverage of and access to the CUDA host APIs from Python, CUDA Python aims to integrate the Python ecosystem [6].

Why Python CUDA?
CUDA Python offers consistent APIs and bindings that may be included in current toolkits and libraries to make GPU-based parallel processing for HPC, data science, and AI is more accessible. Numba, a Python compiler from Anaconda that can compile Python code for execution on CUDA-capable GPUs, offers Python developers

a simple entry into GPU-accelerated computing and a path for using CUDA code with increasingly complex features while introducing the least amount of new syntax and jargon possible. CUDA Python can now take the place of Numba's proprietary CUDA driver API bindings. The speed of a compiled language that is optimized for both CPUs and NVIDIA GPUs is combined with the rapid iterative development capabilities of Python to give you the best of both worlds with CUDA Python and Numba [6].

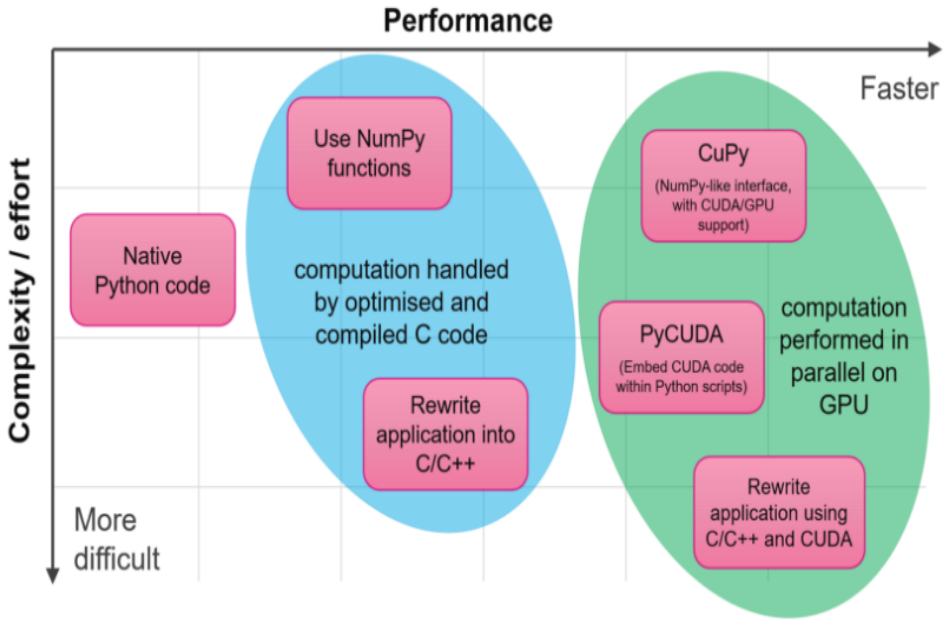

Figure 3.2: Complexity and performance in heterogeneous computing can be traded off, developing own CUDA code can significantly reduce run-time. [95]

CuPy is a NumPy/SciPy compatible Array library for Python GPU acceleration developed by Preferred Networks. When importing the CuPy Python module, CUDA Python speeds up the process and reduces the memory footprint. When more CUDA Toolkit libraries are available in the future; CuPy will require less maintenance and have fewer wheels to release. A quicker CUDA runtime is good for performance. The Python CUDA ecosystem offers comprehensive coverage of and access to the CUDA host APIs from Python using a single set of standard interfaces. The ecosystem is to be built upon in concert with combining various accelerated libraries to address the issues. It also makes it easier for programmers to access NVIDIA GPUs [6].

Python CUDA Workflow:
Python is an interpreted language, so you'll need a mechanism to translate the device code into PTX before you can extract the function that will be used later in the application. Parallel Thread Execution (PTX) is a low-level virtual machine

and instruction set architecture, though it is not necessary to comprehend CUDA Python (ISA). Your device code is created as a string, and NVRTC, a runtime compilation library for CUDA, is used to compile it. Create a CUDA context and the necessary resources on the GPU manually using the NVIDIA Driver API. Then, run the produced CUDA code and get the output from the GPU [7].
Compared to standard general-purpose GPU (GPGPU) computing using graphics APIs, CUDA offers the following benefits [83]:

- Random reads: Code is capable of reading from any address in memory.
- Common virtual memory (CUDA 4.0 and above)
- CUDA presents a quick shared memory area that can be shared by multiple threads. In contrast to texture lookups, this can be utilized as a user-managed cache to enable better bandwidth.
- Improved download and readback speeds to and from the GPU
- Full support for bitwise and integer operations, including lookups for integer textures.
- The CUDA cores are used for a function dubbed "RTX IO" on RTX 20 and 30 series graphics cards, which significantly speeds up loading.

3.3.1 Numba for CUDA GPUs

Numba a just-in-time (JIT) compiler for python, performs best with code that employs NumPy arrays, functions, and loops. The most popular way to use Numba is to apply one of its many decorators to your functions to tell it to compile them. All or a portion of your code can then run at native machine code speed when a call is made to a Numba decorated function since it is "just-in-time" (JIT) compiled to machine code for execution [1].
Numba works with the following [1],

- OS: Windows (32 and 64 bit), OSX, and Linux (32 and 64 bit)
- Architecture: x86, x86-64, ppc64le. Experimental on armv7l, armv8l (aarch64).
- GPUs: Nvidia CUDA. Experimental on AMD ROC.
- CPython
- NumPy 1.10 to the latest

As shown in figure 3.3 Numba must transform an extremely expressive, dynamic language into one that employs very simple, precise instructions tailored for different types. This is accomplished by Numba's pipeline through a series of steps, each of which moves the representation away from the Python source and toward

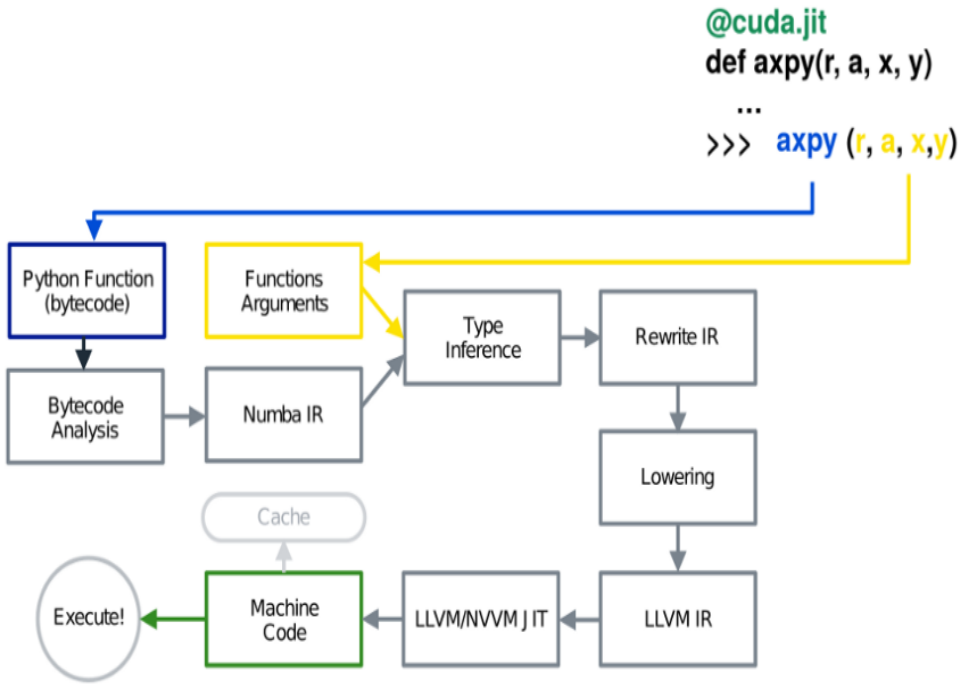

Figure 3.3: Execution pipeline from Numba to Machine code [52]

machine-executable code. A limited fraction of Python code is immediately compiled by Numba into CUDA kernels and device functions in accordance with the CUDA execution model to facilitate CUDA GPU programming. There seems to be direct access to NumPy arrays in kernels developed in Numba. NumPy arrays are automatically moved between the CPU and the GPU. The Python bytecode for a decorated function is read by Numba, and then mixes it with details on the types of the function's input parameters. It then utilizes the LLVM compiler library to create a machine code version of your function that is suited to your CPU capabilities after analyzing and optimizing your code. Then, each time your function is called, this built version is applied [1]. Using the LLVM compiler infrastructure, Numba converts pure Python code into efficient machine code. Without switching languages or Python interpreters, array-oriented and math-intensive Python code can be optimized just-in-time to perform on par with C, C++, and Fortran [2].
Main features of Numba [2]:

- Live code creation (during import or runtime, depending on the user's option)
- CPU and GPU hardware native code generation.
- Numpy interaction with the Python software stack.

3.4 Eye Tracking Data Extraction

Eye tracking is a technology that records and analyzes a user's eye movements and positions. An eye tracker can be used to gather and store eye data. Eye-tracking data provide previously unattainable insights into human behavior and surroundings, digitizing how users interact with computers and opening up new possibilities for passive biometric-based classification techniques like emotion, cognitive prediction and patterns. Reviewing the features and characteristics that can be acquired from eye-tracking data for the classification problem is our goal [48].

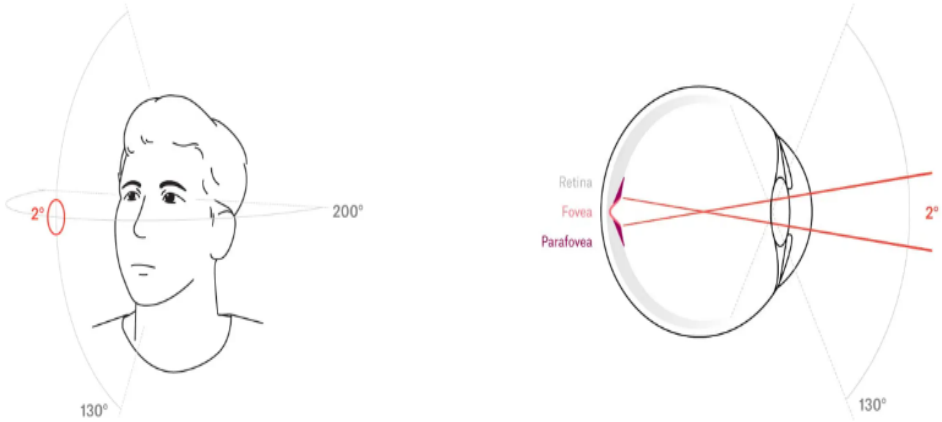

Figure 3.4: Detection of Eye Moment [4]

The interface between artificial intelligence (AI) and human-computer interaction (HCI) is becoming more and more popular currently. Today, more and more important attempts are being made by scientific experts to study unique interactions between humans and technology. Since machine learning can learn automatically and do specified tasks at a human level of proficiency without the assistance of a human expert, studies on classification using machine learning, including emotion prediction and image classification, have gained a lot of popularity. There are therefore studies on classification using various methods, such as picture classification using neural networks and emotion classification utilizing brainwave signals [100], [104]. The work by Nilsson was a sample study on learning machines. However, it focused more on machine learning for pattern classification [59]. In recent years, there have also been more studies on eye tracking. The topic of how eye-tracking data can be used in study drives many researchers' experiments. Therefore, the question of what eye traits may be extracted from eye-tracking data for classification arises when using eye-tracking technology in classification research. Eye-tracking

technology is the method of monitoring and calculating a user's eye movements and focal points. Eye tracking is frequently utilized in a wide range of fields, including psychology, marketing, medicine, video games, and cognitive research. As a result, computer science departments are increasingly using eye tracking, which makes use of eye features to research information processing problems [65]. An eye-tracking sensor or a camera can be used to measure and collect eye-tracking data. The data offer a variety of attributes and can be applied to a variety of classification tasks. As it simply needs a basic camera to capture the necessary data, eye-tracking technology is quite useful, and it can be extensively adopted and applied in the future [48].

3.4.1 Eye-Tracking Technology

An innovative technology called eye-tracking technology is used to track a user's eye movements or point of fixation. It is a technique of determining a person's point of gaze or eye location and gathering information about their ocular characteristics. The results are recorded as data, which includes detailed statistics like fixation counts, first fixations, and fixation length. These captured data can be examined, and the ocular features can be extracted utilizing visual analytic methods. Using a visual analytical technique will help you see typical visual problem-solving techniques better [11]. Data visualization tools like heatmaps and saliency maps can be used to visually explore and evaluate eye data [48].
Fixations, saccades, and scan paths are the three primary ocular activity indicators that can be used to classify eye-tracking data. Fixations, which last between 100 and 400 milliseconds and stabilize the retina above a stationary object of interest are eye movements. The fixations are generally in the middle, and the eye moves slowly. Ocular drifts, ocular microtremor, and microsaccades are their distinguishing features [64]. Saccades are rapid eye movements performed by both eyes to transfer the fovea, the central region of the retina, to a different spot within the visual field. Saccadic motions are reflexive and voluntary, and they normally last between 10 and 100 milliseconds [22].
Predictive saccade, antisaccade, memory-guided saccade, and visually-guided saccades are the four different classifications of saccades [72]. The direction that a viewer's eyes move in when reading a text or taking in a scene is referred to as the scan path of eye movement. The information about the eye's movements while the visual field is scanned, and some type of visual data is evaluated and analyzed is known as scan path data. The resultant series of saccades and fixations is a scan path [48].

Eye-Tracker
An eye tracker is a tool for detecting eye movements and positions. It is designed to track a subject's eye movements as they proceed through a task and pay attention to a stimulus in order to gauge their level of visual attention. Eye-attached

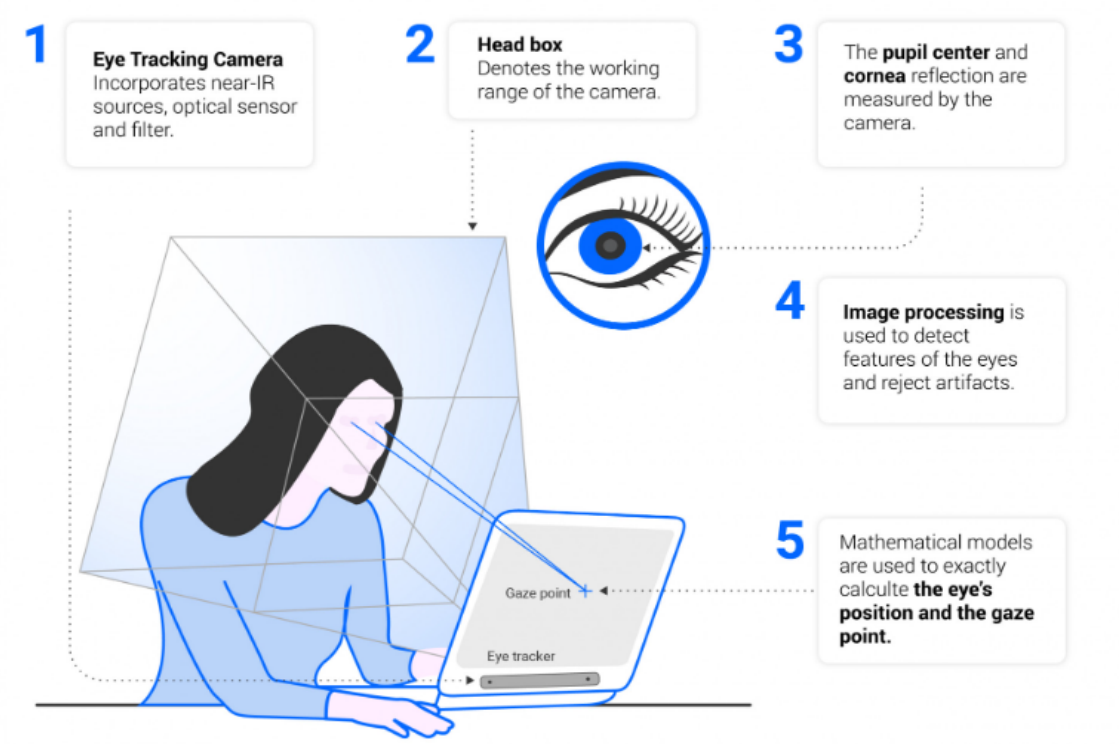

Figure 3.5: Phases involved in Eye Tracking [54]

tracking, optical tracking and measuring electric potentials with electrodes are the three categories of eye trackers. Tracking with the use of an eye attachment, such as a specialized contact lens, is known as eye-attached tracking. On the assumption that it does not move significantly as the eye rotates, the attachment's movement is computed. This technique enables the measurement of the eyes' torsional, vertical, and horizontal eye movement [69]. Optical tracking tracks the positions that are attached to an object and uses that information to identify the position of the object in real-time. Using a camera gadget, the reflex point's location is identified. Without coming into touch with the eye, the optical technique monitors eye movement. This approach is frequently used for gaze tracking, particularly those that involve video capture and is favored since it is less expensive and non-intrusive. The third kind of eye tracker uses electrodes to measure electric potentials. Even with the eyes closed and in perfect darkness, a constant electrical field emanating from the eyeballs can be seen. The Electrooculogram (EOG) is an illustration of this tracking technique [48]. It is a method for calculating the corneo-retinal standing potential, which exists between the forehead and the retina of the eye. It is a very simple approach that uses very few processing resources. It also functions in a variety of lighting situations and can be used as an integrated, standalone wearable gadget [88].
To track gaze direction, the majority of contemporary eye trackers combine near-infrared technology with a high-resolution camera (or other optical sensors). The

basic idea, known as Pupil Center Corneal Reflection (PCCR), is actually rather straightforward. In essence, the camera tracks the corneal reflection of light and the pupil's center. The eye tracker receives information about the movement and direction of the eye from the light reflected from the cornea and the center of the pupil. As shown in figure 3.6 [25].

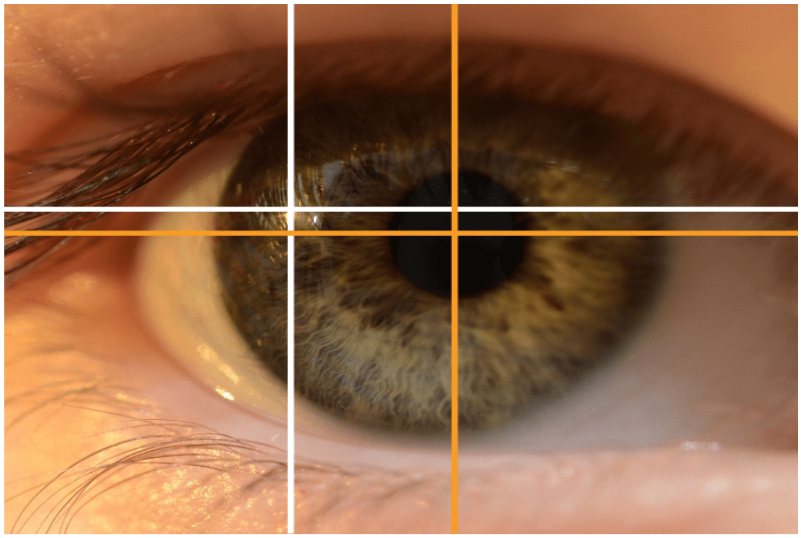

Figure 3.6: Pupil Center Corneal Reflection (PCCR) [25]

Machine Learning

Artificial intelligence (AI) in the form of machine learning (ML) enables computer programs to forecast outcomes more accurately, without having been expressly taught to do so. In order to forecast new output values, machine learning algorithms use historical data as input. The predictions without being specifically programmed to do so, it builds a model using the training data [44]. Precision or accuracy estimation approaches are used in cognitive science to assess the effectiveness of emotion classification using machine learning. The Support Vector Machine (SVM), K-nearest neighbor (KNN) and random forest were the three machine learning methods that were most frequently utilized. Based on the features from datasets, predictions and classification tasks are carried out [43]. In several domains, including computer vision, where the necessary tasks cannot be completed using conventional techniques, machine learning can be used. A scientific study suggested a method for determining a toddler's age by looking at their gaze patterns [21]. Another study uses eye tracking to identify personality traits from outside stimuli [12]. In order to carry out certain tasks, machine learning also entails computer learning from information or data provided [48].

3.4.2 Eye-Tracking Features

Pupil Size

Beyond merely light, the pupils also react to other stimuli. They serve as a cue for arousal, curiosity, or emotional exhaustion. Individual differences in intellect are directly related to the baseline pupil size. According to assessments of reasoning, attention, and memory, the smarter the subject, the bigger the pupils. In fact, the unaided eye was able to distinguish between individuals with the highest cognitive test scores and those with the lowest baseline pupil sizes. The locus coeruleus, a nucleus in the upper brain stem, with extensive neuronal connections to the rest of the brain, is associated with an activity that affects pupil size. Norepinephrine, a neurotransmitter and hormone that controls bodily and mental activities like perception, focus, learning, and memory is released by the locus coeruleus. Additionally, it supports the healthy coordination of brain activity so that various brain regions can cooperate to complete difficult activities and achieve ambitious objectives. In fact, the brain spends the majority of its energy maintaining this structure of activity even when we aren't doing anything at all, such as when we look at a blank computer screen for hours on end [41].

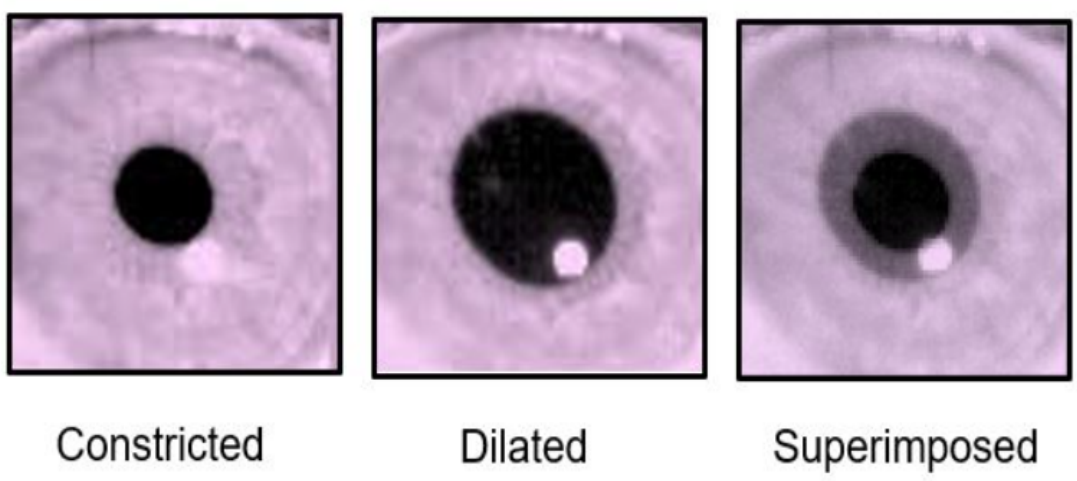

Figure 3.7: Different stages in Pupil Dilation [3]

Fixation and Gaze Points

Fixations and gaze points are the primary output measures of interest and frequently used terminology when discussing eye tracking. What the eyes are gazing at is revealed by gaze points. You will get 60 distinct gaze points per second if your eye tracker samples data at a rate of 60 Hz. A fixation is a period of time when the eyes are locked onto an item and occurs when a cluster of gaze points are very close together in time and/or location. Fixations are good indicators of visual attention, and this area of study has been expanding steadily. Saccades are the common name

for the eye movements that occur in between fixations. For instance, while we read, our eyes don't move easily. Every 7-9 letters, our eyes are locked (although this, of course, depends on the font type and size). How many words we can read before and after the word that is currently fixated is referred to as our "visual span." A trained reader can cover more material with fewer fixations because they have a longer visual attention span. While watching a faraway car pass by, though, our eye motions are very different. Here, we pursue without jerking or saccadic motion. However, if the item is moving too quickly or is too unpredictable, there may be catch-up saccades [5].

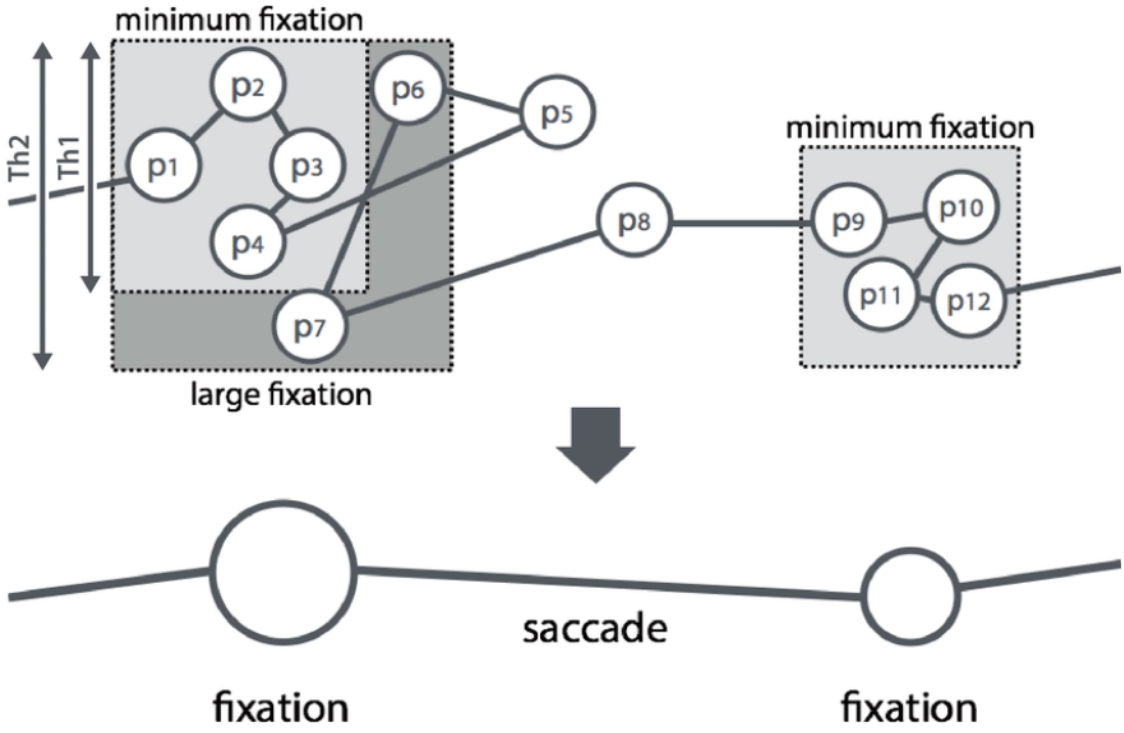

Figure 3.8: Identification of fixations and saccades [66]

Saccade
A quick conjugate eye movement that quickly moves the center of the gaze (line of sight) from one area of the visual field to another is typically employed to turn one's head in the direction of a particular item. It exhibits stereotypical amplitude, duration, and peak velocity correlations. Only saccades are easily performed at will (as when scanning a visual), but they are also deeply entwined with reflexive and involuntary activities. A reflexive saccade toward the position of the stimulus is elicited by suddenly occurring visual, as well as perceptual and sensory, stimuli (visual grab reflex), and the eyes are also involved in an ongoing sequence of microsaccades during seemingly undisturbed fixation [91].

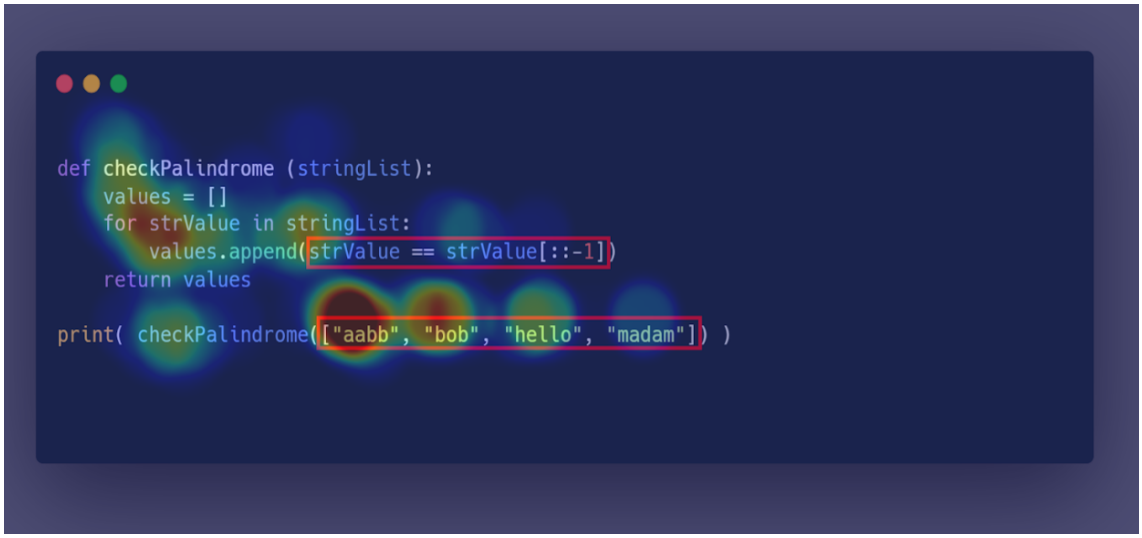

Figure 3.9: Heat map example of palindrome exercise

Heatmaps
Heatmaps are graphic representations of how gaze points are generally distributed. They are often shown as a colored overlay over the image or stimulus that is being delivered. The shades of red, yellow, and green depict, from most to least, the number of gaze points that were focused on various areas of the image. A simple way to rapidly see which elements are more popular than others is by using a heatmap. Heatmaps allow comparisons between individual responses and groups of participants, which can be useful for figuring out how various populations may perceive a stimulus differently [5].

Areas of Interest (AOI)
An area of interest, also known as an AOI, is a tool used to pick out specific areas of a shown stimulus and extract metrics for those areas. It does not constitute a metric in and of itself, but it does define the space on which other metrics are based. You could, for instance, create different AOIs around the torso and the face of a person in a picture. You will then be able to see metrics for each region separately, such as how long it took participants to gaze at a region after the stimulus appeared, how long respondents stayed in a region, how many fixations were recorded, and how many respondents glanced away and back. These metrics are useful for comparing the effectiveness of two or more components of a single video, image, website, or software interface [5].

Time to First Fixation
The Time to First Fixation (TTFF) measures how long it takes a respondent (or all respondents collectively) to fixate on a particular AOI after the stimulus has begun. TTFF can represent both top-down attention-driven searches as well as bottom-up stimulus-driven searches (such as a dazzling brand label) (e.g. when respondents ac-

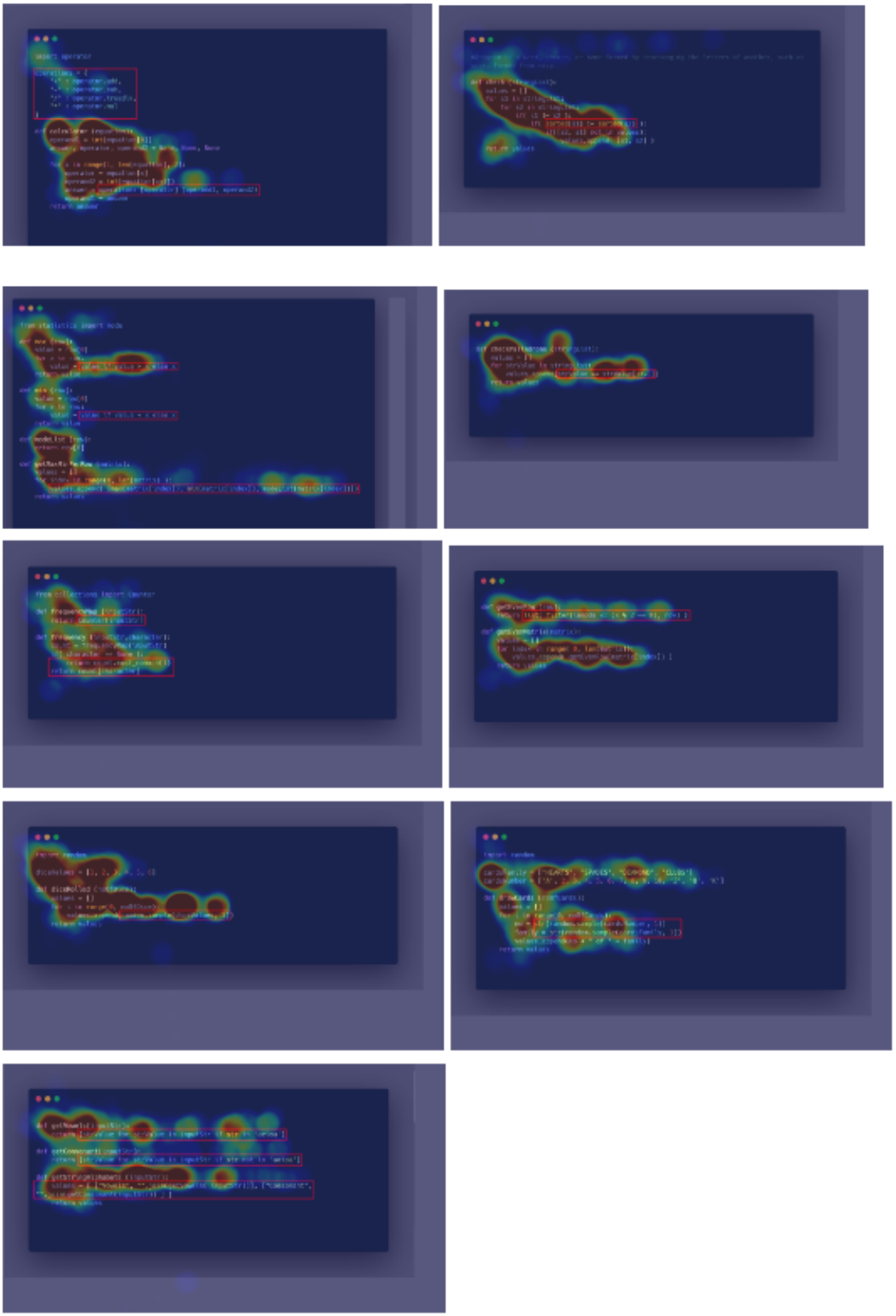

Figure 3.10: Average Heat map of each exercise of trained group

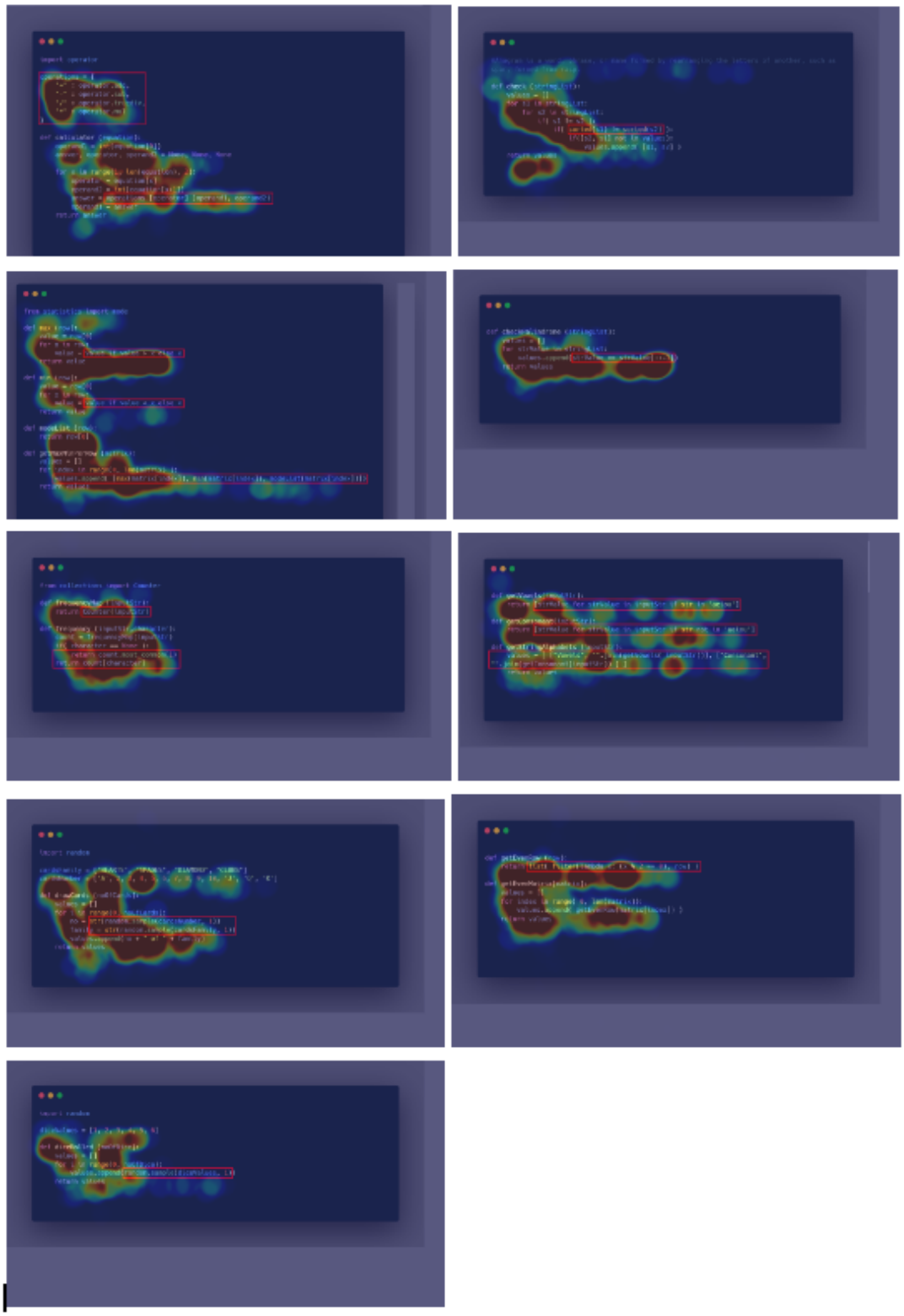

Figure 3.11: Average Heat map of each exercise of untrained group

tively decide to focus on certain elements or aspects of a website or picture). TTFF is a simple but extremely useful eye-tracking metric that can reveal how particular elements of a visual environment are prioritized [5].

Time spent (Dwell Time)
The duration of time respondents spend gazing at a certain AOI is measured as time spent or dwell time. In some circumstances, an increase in the amount of time spent on a particular area of a picture may indicate motivation and top-down attention when participants avoid looking to potentially equally fascinating stimuli in the visual periphery. Shorter duration times can suggest that other regions on the screen or in the environment might be more fascinating, whereas longer duration times can suggest a high level of attention. However, eye tracking alone cannot draw inferences about the emotional response to the visual stimulus (other measures, such as facial expression analysis or EEG can help fill in the gaps) [5].

Revisits
The number of revisits tells us how frequently a participant returned to a specific location that was identified by an AOI. This enables the researcher to investigate which sections consistently drew the participant (for better or worse) and which ones they saw but quickly abandoned. The person could be drawn to a certain aspect of an image because it is interesting, difficult to understand, or even frustrating. Eye tracking can give you information about what needs more investigation, even though it cannot tell you how someone felt while looking at something [5].

Scan Path

Scan Paths are based on the timing and location of participant looks, as well as spatial and temporal information. This makes it possible to build up a picture of what a participant prioritizes when they observe a visual scene. Due to the central fixation bias, this will frequently start in the middle of the image, but the subsequent viewed elements will be an accurate representation of what the participants are most motivated to look at. Since the order of attention reflects both a person's interest and the most important aspects of the display or surroundings, it is frequently employed as a marker in eye tracking studies (i.e., elements that stand out in terms of brightness, hue, saturation, etc.). The last fixation is more likely to predict a choice, but there is a caveat: this fixation frequently occurs when experimenters are aware of the top-to-bottom and left-to-right reading biases. Most languages have a top-left-to-bottom-right reading order. First fixations are, therefore not particularly indicative of behavior because items in the visual fields are frequently randomized throughout repeated trials (due to this manipulation). In the actual world, it appears that smart marketers and designers will frequently be aware of these visual biases, allowing them to possibly influence situations so that first fixation count [5].

First Fixation Duration

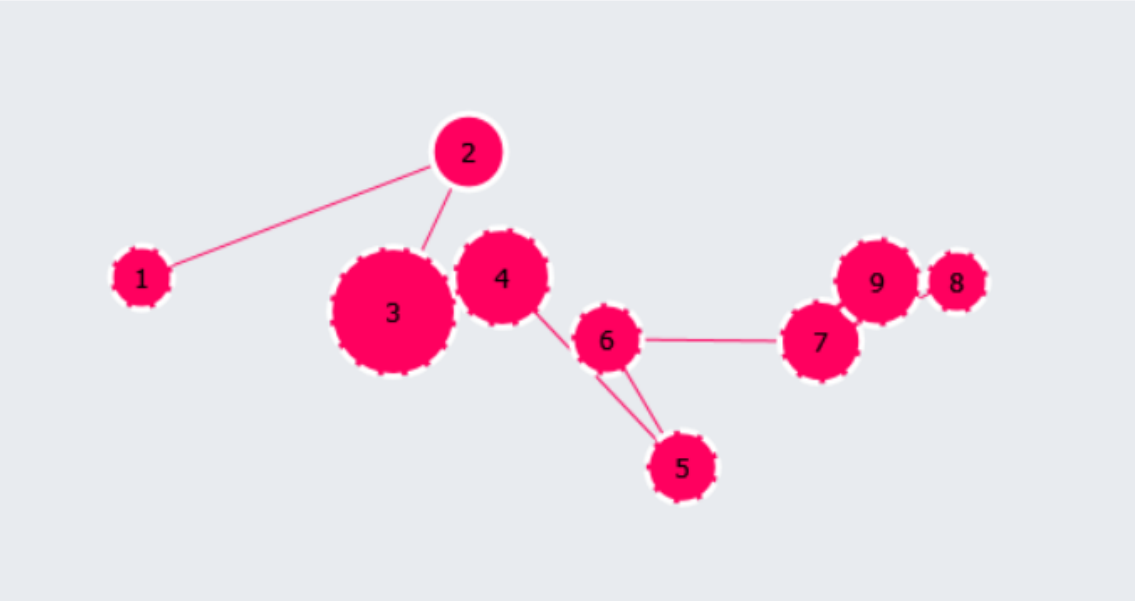

Figure 3.12: Scan Path Example [51]

Our eyes travel around a visual scene as we investigate it before fixating on a particular area of the image. Data on the length of time that the initial fixation lasted is provided by the first fixation duration. When combined with TTFF, this can be very helpful in determining how much a certain feature of the scene initially caught the audience's attention. The location is probably particularly eye-catching if a participant has a low TTFF and long first fixation duration. This information, collected from an AOI, is especially helpful since it shows how long the first fixation lasted in a particular location in comparison to other regions. This can be helpful in figuring out what an AOI's initial impressions are [5].

Average Fixation Duration
The average fixation duration, which can be calculated for both individuals and groups, provides information on how long the average fixation lasted. In either situation, this can be useful as a baseline measurement, but it's also intriguing to consider across stimuli. If one image results in a much longer average fixation duration than another, it can be worthwhile to investigate the causes. Additionally, comparisons between AOIs enable you to identify the places that were actually given greater attention than others. It's likely that you'll want the average fixation duration to be longer in the parts that showcase the message than in other areas if you're trying to communicate a message [5].

Blink
The majority of the time, blinking is an instinctive closing and opening of the eyelids. Raw data points lack the x, y coordinates information because the eyelid covers the pupil and cornea during each blink, blocking them from the illuminator. Fixation

filters can be applied during the analysis to eliminate these points and accurately extrapolate the data into fixations. It is also possible to extract information on blinks from the raw data gathered by the eye tracker, provided that the head motions are within the eye tracker requirements, i.e., that the missing data points do not result from moving the head away from the eye tracking box [8].

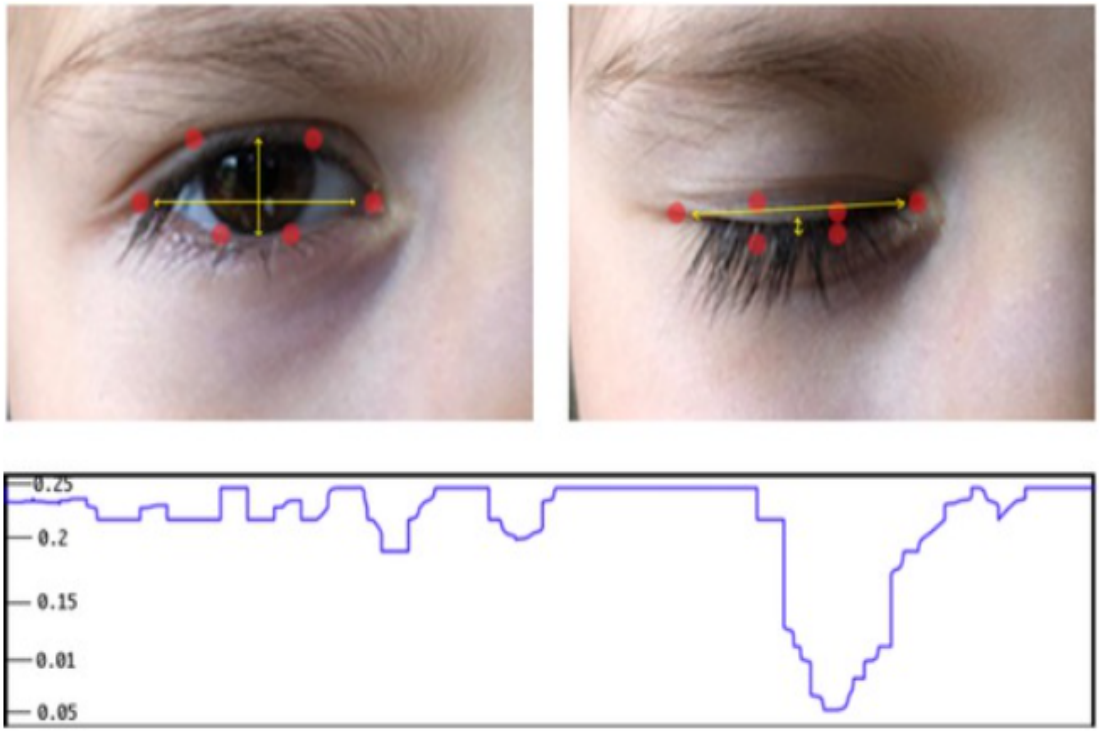

Figure 3.13: When the blink is detected the respective pupil dilation value is set close to zero [71]

4 Design

4.1 Objective

In this paper, our objective of work is to present some insights into whether students are able to comprehend shortcut code. To accomplish our task, we examined how the shortcuts affected students' behavior and visual attention. Further, to gain more insights, we included trained a group of students and then categorized them into two separate groups.
To add clarification to our objective, we address the research questions mentioned in the next section.

Research Questions
While comprehending the source code:

1. Do program shortcuts affect students' response time and correctness?

2. Do program shortcuts affect students' visual attention?

Relevance
The motivation for us is to find out whether there is any significant difference between understanding special syntax or shortcuts used in any code amongst different experienced groups. It also provides us insights into their knowledge of the programming language and problem-solving skills. On the other hand, if the shortcuts are taught to a group of students, can they comprehend the code and how well they could perform using those shortcuts in different scenarios to achieve different results?

Method
Each participant grasped the fundamentals of Python. Each participant will see the snippets in order, and behavioral and visual information was captured that will be crucial to the research and findings. 11 out of the total 21 participants took the pretest to become familiar with the shortcuts. During the experiment, we used a Tobii Pro Fusion eye tracking equipment to monitor the user's eye movement. Students were asked to select the correct response from a list of four possibilities after each exercise in the study, which was conducted using PyschoPy. At the conclusion of each test, behavioral and correctness data were generated which was used in analysis and findings.

Implications

The basic objective is to determine whether there are any appreciable differences in how different experienced groups comprehend shortcuts (special syntax) used in any code. It also gives us information about their programming knowledge and problem-solving abilities. On the other side, if a group of students are taught the shortcuts, do they understand the code and how effectively they could use those shortcuts in various contexts to get various results.

4.2 Experiment Procedures

The experiment included the data listed in table 4.1. Each participant knew the fundamentals of Python. Every participant saw the snippets in order, and behavioral and visual data were collected as per table 4.2, which was crucial to the analysis and outcomes. There were 21 individuals in all, and 11 of them participated in the pretest, learning about the shortcuts. In the study, we used a Tobii Pro Fusion eye tracking device to track the eyeball movement of the user. For the students to participate, PsychoPy gives them an interface to view the source code. The source code displayed using PsychoPy helped to record their eyeball movement. In the beginning, students are shown code snippets with a source code; on the next page, inputs are added to the source code. Students are then requested to select a correct answer from the four available options. Response time, correctness, post-talk-aloud, experience, and visual attention are analyzed as the dependent variable. Response time defines the amount of time taken by the participant when they start viewing a snippet until the time when they submit the answers. At the same time, correctness refers to answers submitted by the participant after the execution of the print statement of the source code. (e.g., check palindrome in the above figure). The post-talk-aloud session was conducted, where the audio between the researcher and participants was recorded. The participants were asked to share their responses regarding the difficulty of the snippets. The questions were based on the heatmap generated by the visual data for every snippet and if they could identify the shortcut used in the code.

4.3 Software and Hardware

4.3.1 Psycho Py

Version: 1.5
PsychoPy (Open-source software) is largely used in experiments in neuroscience and psychology. It was initially created as a Python library, then as a graphical application, and it now also supports JavaScript outputs to run studies on the web and on mobile devices. In contrast to other packages, it offers a user interface, allowing them to create experiments using graphical interfaces or Python scripts or even by combining them both. It achieves platform neutrality by leveraging OpenGL for graphics calls and the wxPython widget library for the application. Additionally, it

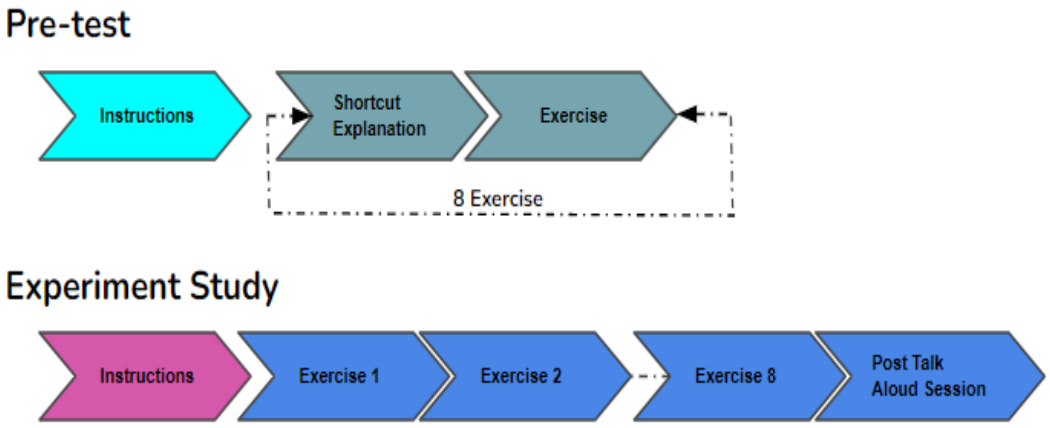

Figure 4.1: Experiment Design Visualization

also has the ability to produce and transmit audio impulses. The program PsychoPy is excellent for designing experiments. It blends Python programming's elegance and power with a graphical user experience. It has clearly encouraged greater integration of psychology with neuroscience because it has established itself as the preferred open-source language for multidisciplinary initiatives in neuroscience. Since it is open source, it is also a powerful alternative for MATLAB to carry out neuroscience and psychology experiments. The PsychoPy community keeps assisting many novice researchers with experiment programming. It can still be expanded indefinitely to accomplish almost any scientific objective. The experiment involved students' participation by using PsychoPy. The results of the experiment produced were very accurate and valuable.

4.3.2 Tobi Pro Fusion

With the inclusion of Eye tracking in the experiment, it encouraged us to explore further in-depth various questions, which led us to discover new ways to approach scientific inquiries. Besides adding value and developing observation metrics, this motivated us to alter these methods of collecting data from the participants. The easy-to-use, compactness of the device makes it an ideal entry point to begin their research or wish to take the research to the lab environment. Tobii pro Fusion, embedded with two cameras, one for stereoscopic eye tracking and another for integrated for processing for dark and bright pupil mode. The information acquired from the device offers us more details about the participants' visual data to assist in observing their behavior and determine the ways to make the experiments better in the future. It also provides the flexibility to modify your data according to the requirements, offering a sampling frequency of up to 250Hz. Detailed information with sampling rates up to 250 Hz there is more tolerance observed for head movement.

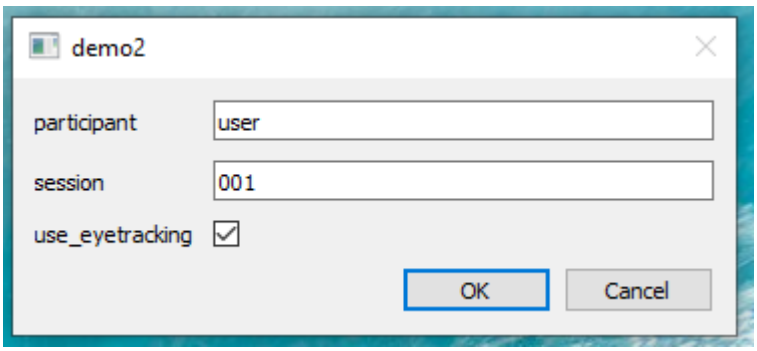

Figure 4.2: Initial dialog of Pyscho Py Study

Timing consistency is ensured because there is no need for an external processing unit because the data is processed independently.

4.3.3 Setup and Configuration

After the consent of the participant the configuration was set up. Firstly, the calibration of the Tobii pro fusion eye tracking device with the participant was set. After calibration, the PyschoPy study was executed with the initial prompt of the dialog as per figure 4.2 where the information about the user number and session was filled and the "use_eyetracking" checkbox was enabled. Later, the instruction and rules were followed. The instruction stated there will be some basic python code fragment few of them are formal logics and lesser mathematical function. Some code structure and function names can be bit unusual, but they do not need to bother about that and solve the tasks as best as they can. Their task was to comprehend the code snippet's which will be present to them. After that they have to choose the right option for the final print statement from a possible output displayed to them. Spacebar can be used to go to the next page of the study.

4.4 Materials (Data)

Based on the discussion in section 3.2, nine code snippets were chosen. Shortcut codes, such as logical or syntactic, are used in the code snippets, which are listed in Table 4.1 and illustrated in Image 4.3. The study featured 55.6% logical shortcut codes and 44.4% of syntactic shortcut codes. The snippets were chosen to examine and gauge how participants engaged and looked at the shortcut code fragment and how long it took them to fully understand it. The dimension of all the code snippets images were 1080x1920. In the post-study, the participants were asked to give us an idea of the level of complexity of each piece of code. We graded the difficulty of the code snippets from easy to medium after observation. The majority of the participants had no trouble recognizing the shortcuts. We proved that individuals

Figure 4.3: Syntax vs Logical shortcuts used in the study where **syntax shortcuts** are the ones which are checked at compile time and **logical shortcuts** are the combination of one or more semantic(s) logic which performs one or multiple operations in one particular statement.

who took part in the pretest and had prior Python competence were able to use that experience to finish the experiment study.

The experimental investigation used two different versions of the code samples.

Shortcut Snippet	Lines of Code (LOC)	Cognitive Weights (Wc)
Basic Calculator	22	11
Anagram	12	28
Max Min from a Matrix	22	37
Palindrome	7	13
Alphabets Frequency	14	2
Get Even Elements	12	22
Random Dice	12	11
Random Card	15	11
Vowels and Consonants	14	22

Table 4.1: Code Snippets list with LOC and Cognitive Weights

Figures 4.4 and 4.5 show the difference; 4.4 has an actual code fragment, and 4.5

retains the same combined with an enlarged code fragment and a final print statement that is then used to help the participant choose an answer from the available options to assess their performance.

```
import random

diceValues = [1, 2, 3, 4, 5, 6]

def diceRolled (noOfDice):
    values = []
    for i in range(0, noOfDice):
        values.append(random.sample(diceValues, 1))
    return values
```

Figure 4.4: Example of the of Code snippets without input statement

```
import random

diceValues = [1, 2, 3, 4, 5, 6]

def diceRolled (noOfDice):
    values = []
    for i in range(0, noOfDice):
        values.append(random.sample(diceValues, 1))
    return values

noOfDice = 4
print( diceRolled(noOfDice) )
```

Figure 4.5: Example of the of Code snippets without input statement

4.5 Independent Variables

In this study we have one independent variable as **Code Snippets** as they are constant attribute all over the study for every participant. A code snippet is a brief section of source code or a code fragment. We used snippets of the Python programming language that were comparable for both groups and had shortcut syntax to complete a certain task. The code samples are produced using the online tool Carbon, which enables us to produce source code pictures that resemble code editor panels. Shortcut statements are based on special syntax or logical operations

where **syntax shortcuts** are the ones which are checked at compile time and **logical shortcuts** are the combination of one or more semantic(s) logic which performs one or multiple operations in one particular statement. In addition to categorizing the shortcuts themselves, we incorporate them into brief pieces of code that carry out particular tasks. This can be used by functions, subroutines, procedures, library functions, or algorithms. The example shown below uses both logical and syntactic shortcuts. For more details 4.4

4.6 Dependent Variables

Data received at the end of the study contains information regarding eye tracking (supplied by Tobii Pro Fusion), response time, and correctness since PsychoPy is internally connected to Tobii Pro Fusion. The dependent variable in this study are **response time, correctness, post-talk-aloud, experience, and visual attention**. Response time is the period of time when a participant begins to examine a snippet till they submit their responses. Correctness likewise refers to responses provided by the participant following the execution of the source code's print statement. (See the illustration 3.9 above palindrome example) The post-talk-aloud session was held, and the researcher and participants' conversations were audio-taped. The participants were asked to provide their opinions on how challenging the snippets were and what the tricky part or shortcut involved in the code statements. The questions were based on the heatmap created by the visual data for each code snippet and whether or not the participant could recognize the shortcut employed. Next, the visual attention data were retrieved, as described in the section 3.4

Behavioral Data	Visual Data
Response Time [s]	No of Fixations in AOI
Correctness	Duration of Fixation in AOI [s]
Talk Aloud Data	First Pass Duration [s]
	Second Pass Duration [s]
	Revisits

Table 4.2: Different Behavioral and Visual Data in our study

4.7 Participants

We invited students enrolled in undergraduate and graduated degrees at the Chemnitz University of Technology as well as a few experienced individuals with some work experience. All participants had a basic knowledge of Python (i.e., expe-

rienced, passed, at least an introductory programming class). Table 4.3 gives a summary of the demographics and programming background of our participants.

Demographic Data	(years)
Male	13
Female	8
Age	26 ± 3.4
Learning Programming (in years)	4 ± 2.7
Python Programming (in years)	2.3 ± 1.9
Professional Programming (in years)	1.1 ± 1.1

Table 4.3: Demographic Table

4.8 Task

4.8.1 Pre-Test

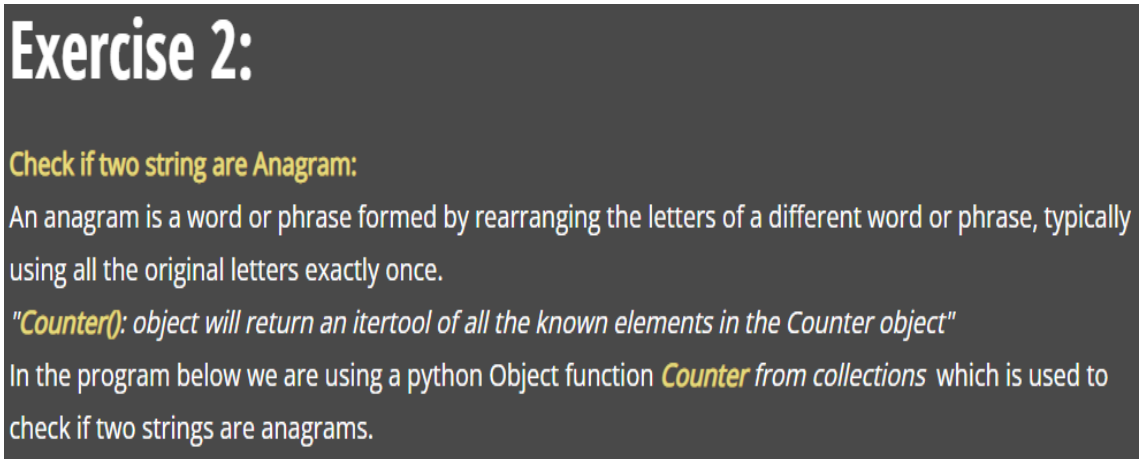

Figure 4.6: Pretest: Explanation of the Shortcut statement

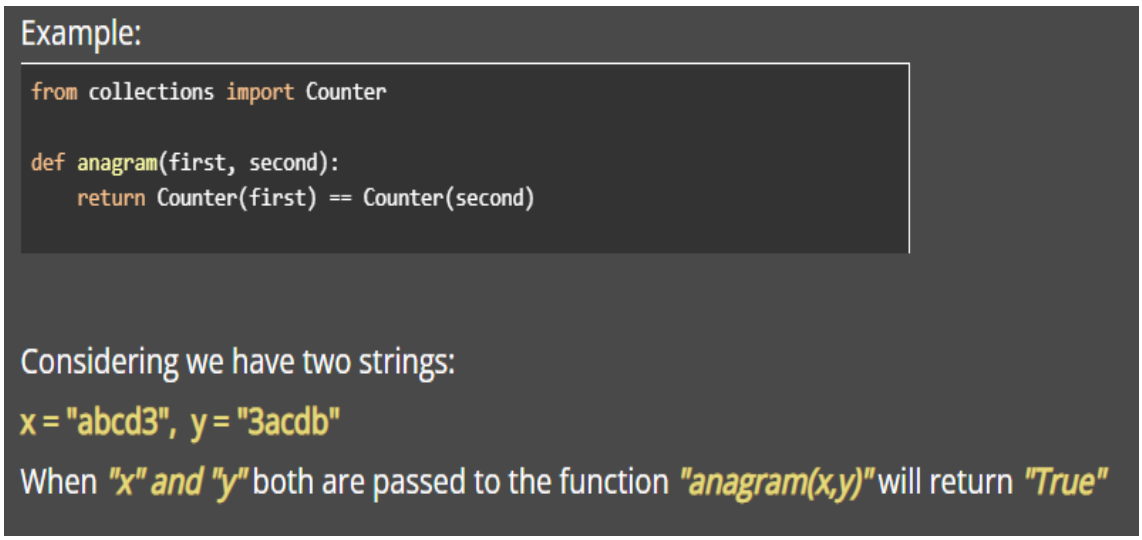

Figure 4.7: Pretest: Example task of the Shortcut

We utilized free online surveys to provide users with a platform to conduct pretests. The participants took a pretest to divide into groups and educate about the shortcuts that would be used in the experimental investigation. Eight tasks in all, each including an explanation, an example task, and an exercise, were offered to the participants, as illustrated in the image below, which shows an example of a task that was part of the pretest. These eight tasks were adequate for participants to understand and move on to the experimental study. Those who took part in the pretest were grouped to the trained group. Figure 4.6 is an example that unveils the pattern we followed to construct the structure to execute the task. The top section provides the description and explanation for the shortcut. As shown in the figure 4.6 shows an exercise for checking if two strings are Anagram together with a description of what an anagram is and how we can achieve the results. In this example, we used Counter from the collection, which returns an element list with a number of occurrences which will assist in finding the frequency of alphabets in both strings and further compare the frequency of every character to check whether both of the strings are anagrams of each other. Figure 4.7 puts up a demonstration of the use of the Counter object to check the anagrams with a code snippet and question. Furthermore, there were three tasks that a participant had to solve with the use of the similar Counter object as shown in figure 4.8. To proceed with the new exercise, the participant must submit their answer.

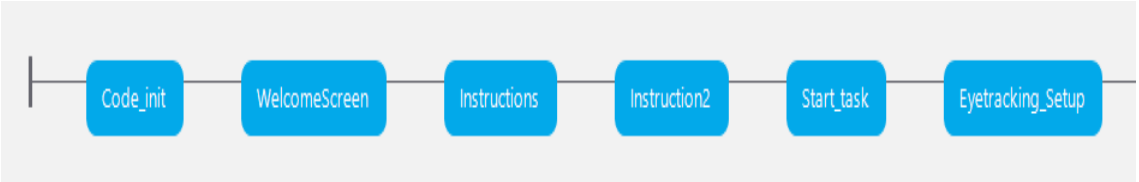

Figure 4.9: Initial flow of Pyscho Py Experiment

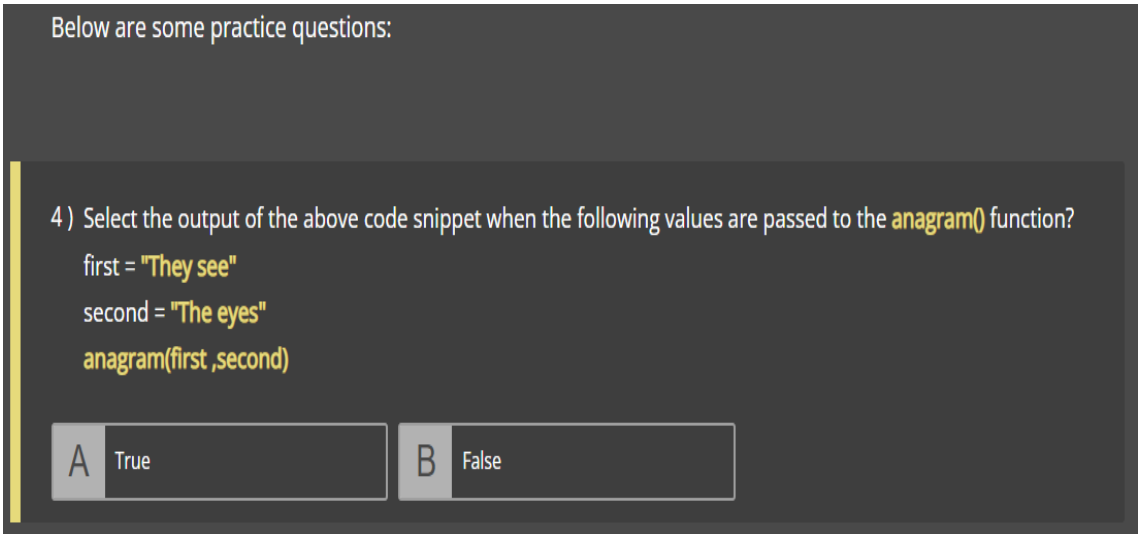

Figure 4.8: Pretest: Participants exercise of the Shortcut

After undergoing 24 tasks, the participant was required to answer a minimum of 18 approximate questions. A total of 11 participants who participated in the pretest were placed in a trained group according to their performance.

4.8.2 Experiment Task

This section will go over the crucial steps, flow, and data extraction of the psychological investigation of the PyschoPy experiment in depth. Figure 4.9 displays the experiment flow, which is followed by Figures 4.10 and 4.11.
The study was carried out at the Chemnitz University of Technology. Each participant received a consent form for data privacy before the study began, and information was shared that the data were recorded anonymously. All of the preconfiguration procedures were carried out before the experiment research even began to ensure that the data was accurately recorded. All the data was exported in excel at the conclusion of the study, and a post-talk-aloud session was held to document the experiences and challenges users had while taking the test. This gave us further insight into the activity and how active participants engaged in the study. The sections below go into great detail about all of this.

Code_init

We determine the flow of the code snippets by accessing the excel config file, which contains information about the code snippets. To begin with, we initialized the eye tracker, which was calibrated for the users to capture the values via eye moments and the variables which will be essential during the process. study_answers and study_cutting_data were two files generated to accumulate the data in response to the action performed by the participants. Time taken by the user to finish one task and answers selected during the process was recorded in the excel files.
Later, the guidelines and norms were adhered to. According to the instructions, there would be some fundamental Python code pieces, some of which would deal with formal logic and simpler mathematical operations. Some function names and code structures may be a little strange, but they don't need to worry about that; instead, they should focus on doing their best to complete the tasks. It was their responsibility to understand the code samples that would be given to them. Then, from a list of potential outputs that are then given to them, they must select the best option for the final print statement. To continue to the study's next page, press the spacebar.

Eyetracking_Setup
Checking whether the use_eyetracking flag is enabled is the first step (Figure 4.2). If so, we start recording the data obtained from the eye tracker that will eventually be exported as an excel file. The recorded values contain gaze point validity, a set of x and y gaze points together with a system time stamp, and pupil diameter (for both the left and right eye).

With the exception of End_Eyetracking, every phase in the flow diagram will loop through nine exercises, and at the conclusion of each phase, the time stamp will be recorded in the study_cutting_data

UpdateIndex
This stage is in charge of iterating through the exercises and gathering input from the next exercise's configured values.

Task and Input
This stage will locate the code snippet's image file from the configuration files and display it to the user. The following page of the task consists of the concrete code fragment in the first image (4.4) and the same input coupled with an enlargement of the code fragment with a final print statement (4.5) for which the participant must choose an answer.

Choose_Output
The user could choose one of four alternatives that were given to them based on the configuration file's four possible responses. They had to choose one, and it was necessary to do so in order to move on to the next task. Following the selection of the appropriate response from the given options, the responses were added to the

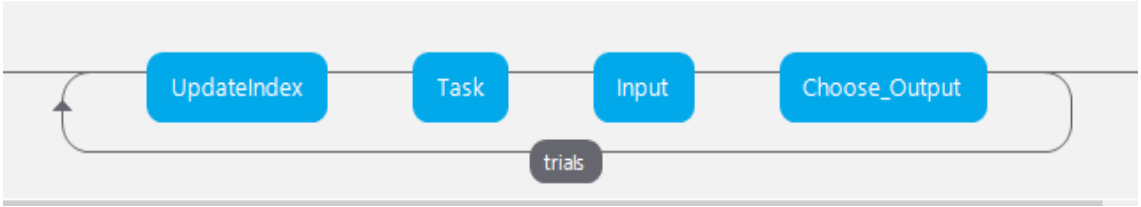

Figure 4.10: Looping through the Code Snippets in Pyscho Py Experiment

Figure 4.11: End flow of Pyscho Py Experiment

study_answers data.

End_Eyetracking
The last phase describes the exportation of study_cutting_data with time values for each exercise, study_answers with user-selected answers for each exercise, and eye tracking data with all the eye tracking data. Eye tracking will be released, and the experiment will be concluded.

4.9 Post Talk Aloud Questions

To comprehend and justify participant decisions, post-experimental questionnaires are becoming more and more crucial in experimental research. However, the choice of how to administer these questionnaires is largely left to the discretion of the individual researcher. It also depends upon analyzing the performance of all-left or all-right item answers, count the number of times a participant chooses an alternative that is rarely chosen by the typical participant, or examine the internal consistency of answer pairs. The information we gather from administering the questionnaire supports our claim that our index does, in fact, evaluate answer quality. Additionally, we identify those individuals whose responses ought to be disregarded as invalid and, as a result, left out of future research by replicating a random-error benchmark [16].
We outline the several careless answer measurement scales now in use, which will be used to create our composite index. We next go into detail on how the questionnaire was put together so that the sample's answers could be evaluated for quality. After creating the questionnaire, we run several experimental procedures. The order of the participants in these procedures varies. The right question can be useful to compare

more detailed results in our hypothesis. Refer to section 4.10 for more details and procedures [16].

4.10 Post Processing

After completion of the study, which included eye-tracking data, answers, duration of completion of the task, and talks aloud data, it was now required to extract meaningful information.

4.10.1 Behavioural Data

As explained before, response time, correctness, and talk-aloud together constitute behavioral data. The inclusion of the Talk-aloud session itself shows its importance because it helped to identify outliers and which participants had actively participated in the study, excluding their contribution towards results and analysis. Response time was evaluated after comparing the end time to the start time. Data with stored answers received from the study were compared to the correct answers, which were assigned with 1 for correct and 0 for incorrect for each task.

4.10.2 Visual Data

Eyetracking data was collected from the beginning till the end of the study for every individual participant. The timestamp column in the data was compared with the duration obtained after the end of the study. For every individual exercise, the entire eye tracking data from the start time till the end time was accessed, which was requisite to construct visual diagrams and feature data. The values contained in the eye-tracking data are shown in table 5.7 and 5.6.
Extracted the eye-tracking values by following the steps.

- Exclude values where right and/or left eye(s) are non-valid.
- Split the set of (x,y) coordinate values x value and y values of both right and left eye.
- Combine the x coordinate values of right and left eye and the same for y values.
- Zips both the x and y together in Gaze values.

Following the steps of preprocessing data, which included cleaning, removing outliers, and further preparing the data for results. Now the values can be passed in order to generate heatmaps.
To further get more insights into the feature values, values of AOI coordinates were created, which proved to be useful in obtaining eye-tracking feature values.

The formula used to calculate fixations and generate visual diagrams (heatmap, fixation, scan path)

$$D = \sqrt{(max(G_x) - min(G_x))^2 + (max(G_y) - min(G_y))^2}$$

$$D = Dispersion\ data\ point$$
$$G_x, G_y = Gaze\ point\ x\ and\ y\ coordinate$$

1. Constructed Heatmaps with a radius of 50 pixels and the sampling frequency of 1000Hz, which were implemented using numba cuda to achieve fast performance.

2. Produced a Fixation diagram of 25 pixels and a minimum duration of 200 milliseconds using PyGaze. (Any duration greater than 200 milliseconds results in data lose).

3. Produced Saccades diagram with a minimum length of saccades of 5 milliseconds, maximum velocity threshold in pixels/second of 40, and maximum acceleration threshold in $pixels/(second)^2$ of 340 using PyGaze.

4. Through modifying of PyGaze fixation generation function, which helped in the extraction of the fixation points and duration of fixations.

5. After receiving all the fixation points, fixation points were compared with the AOI cooridnates, which assisted in obtaining 'FirstPassDuration', 'SecondPassDuration', 'noOfFixtionsInAOI', 'TotalDurationFixationInAOI', 'Revisits', and 'TotalDurationInAOI'.

After the data has been retrieved, preprocessed, and after all subsequent stages have been completed, it is possible to gain considerably deeper insights. The trained and untrained groups were established as described earlier in the study, and the mean value per snippet was determined while eliminating the outlier found in the talk aloud data and correctness (5.2.1). Now that the final data collection has been completed, it is noise-free without outliers, which is then prepared for analysis. This might pave the way for unexpected discoveries and issues. Furthermore, when we compare these groups, we can see various trends.

5 Analysis and Result

5.1 Statistical tests

In addition to your study design, the features of your data will determine the appropriate statistical test to perform. This is the outcome of the study questions and hypotheses we are attempting to address. Data is at the center of statistics. Data by itself is not captivating. We are interested in the interpretation of the data. Statistical testing is a crucial component of statistics. If statistics is defined as the interpretation of facts, then statistical testing can be thought of as the formal technique for examining our views about the world. In other words, we must rely on hypothesis testing if we want to make assertions about the distribution of data or whether one set of results differs from a set of results [36].

5.1.1 T-test

The average values of two data sets are compared using a t-test to ascertain whether they represent the same population. A sample from the trained group in the instances and a sample from the untrained group are unlikely to have the same mean and standard deviation. Similar to how samples from the drug-prescribed group and those from the control group that received a placebo should have slightly different means and standard deviations [36]. The problem statement is established mathematically by using a sample from each of the two sets in the t-test. It presupposes that the two means are equal, which is the null hypothesis. Values are computed and compared to the standard values using the formulas. Accordingly, the presumptive null hypothesis is either accepted or rejected. If the null hypothesis can be ruled out, it means that the data readings are significant and almost certainly not random. One of several tests used for this purpose is the t-test. To evaluate more variables and larger sample sizes, statisticians use tests than the t-test. A z-test is employed by statisticians for high sample sizes [36].

$$t = \frac{m-\mu}{s/\sqrt{n}}$$
$$t = Student's\ t-test$$
$$m = mean$$
$$\mu = theoretical\ value$$
$$s = standard\ deviation$$
$$n = variable\ set\ size$$

By comparing two mutually incompatible statements about a population, we can use hypothesis testing to try to understand or draw inferences about the population.

The goal is to identify which of the two statements the sample data most strongly supports [36].

5.1.2 Outlier Detection Techniques (ODT)

A dataset's abnormal observations/samples that do not fit its typical/normal statistical distribution is found using an outlier detection technique (ODT). Simple approaches for outlier detection examine each distinct attribute of the dataset using statistical tools like box-plot and Z-score. A box plot is a standardized method of utilizing boxes and whiskers to illustrate the distributions of samples matching different attributes. Any data point or sample outside these bounds is regarded as an outlier. The boxes reflect the interquartile range of the data, and the whiskers represent a multiple of the first and third quartiles of the variable [81]. Outliers are often defined as Z-score values more or less than $\pm n\delta$ correspondingly.

$$Z - score = \frac{x_i - \bar{x}}{\delta} \dots (1)$$

Z-score can be explained as equation (1), where x_i is the value of the feature x for the ith sample, $\delta \, and \, \bar{x}$ are the feature x's standard deviation and mean, respectively, of the distribution [81].

5.1.3 Correlation factor

To gauge how closely two variables are related to one another, correlation coefficients are used. The most common correlation coefficient is Pearson's, though there are other varieties as well. The correlation coefficient known as Pearson's correlation, sometimes known as Pearson's R, is frequently employed in linear regression. The degree of correlation between two pieces of data indicates how closely they are related. Pearson Correlation is the most often used correlational statistics metric. The Pearson Product Moment Correlation is its full name (PPMC) [32]. The PPMC is unable to distinguish between dependent and independent factors. A high correlation might be discovered, for instance, if you were looking for a link between a high-calorie diet and diabetes. With the variables reversed, you might, nevertheless, still achieve the same outcome. Alternatively, you may assert that a high-calorie diet results from diabetes. It's clear that this is absurd. The data you are plugging in must therefore be understood by you as a researcher. The PPMC just tells you whether there is a link; it does not provide any information regarding the slope of the line [32].

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \dots (2)$$

$$r = correlation \, coefficient$$

$$x_i = values \, of \, the \, x - variable \, in \, a \, sample$$

$$\bar{x} = mean \, of \, the \, values \, of \, the \, x - variable$$

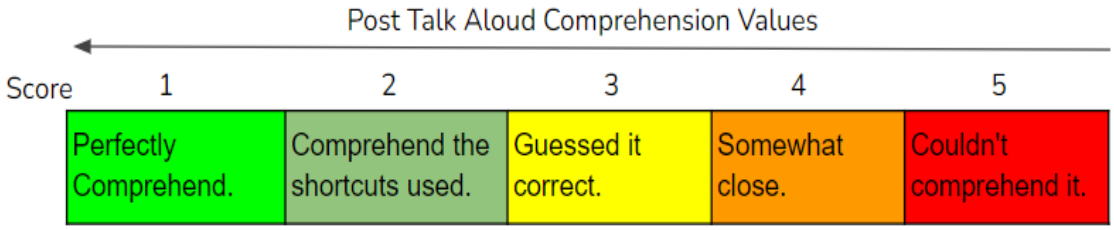

Figure 5.1: Post Talk Aloud performance evaluation

$$y_i = values\ of\ the\ y - variable\ in\ a\ sample$$
$$\bar{y} = mean\ of\ the\ values\ of\ the\ y - variable$$

Equation (2) explains Pearson's Correlation. A value between -1 and +1 is returned by Pearson's Correlation Coefficient, a linear correlation coefficient. A high negative correlation is indicated by a -1, and a strong positive correlation is shown by a +1. A 0 denotes the absence of a correlation (this is also called zero correlation). The measure, as covariance itself, can only account for linear correlations between variables and ignores numerous other kinds of connections or correlations. As a straightforward illustration, one would anticipate a sample of high school student's ages and heights to have a Pearson correlation value that is significantly higher than 0, but less than 1. (as 1 would represent a perfect correlation) [32].

5.2 RQ1: Behavioral Data

5.2.1 Outlier Detection

The mean score for talk aloud and correctness of every participant is shown in table 5.1. Figure 5.1 provides an illustration of how post-talk-aloud scores are evaluated. According to their performances, they were rewarded for their efforts with marks where one was the highest and five was the lowest, and the participants that didn't actively participate in the exercise were considered outliers. After evaluation, three participants' data were pulled out and categorized as outliers and cannot be included in further analysis. Figure 5.2 shows the normal distribution of the post-talk aloud score for the participants calculated through Z-score where,

$$Mean\ (\mu) = 1.84$$
$$Standard\ deviation\ (\delta) = 0.58$$
$$0.99 < (\mu) < 2.69$$

Considering all the values between 0.99 and 2.69, which lie within a standard deviation of ± 1.3 and remain to be considered as outliers. Values of user9, user10, user20, and user21 were recorded far away from the mean. The position of user9 is at the

Group	User	Correctness	TalkAloud
Trained	user1	6	1.7
	user3	3	1.7
	user4	7	1.5
	user8	8	2.1
	user10	2	3.5
	user12	6	2.1
	user13	5	2.3
	user14	6	1.6
	user15	6	1.9
	user16	5	1.9
	user21	5	3.2
Untrained	user2	8	1.4
	user5	8	1.5
	user6	7	1.5
	user7	8	1.4
	user9	9	1
	user11	4	1.6
	user18	4	1.3
	user20	9	2.1
	user21	2	1.9

Table 5.1: Mean score for Talk aloud and Correctness for every participant in trained and untrained group

extreme right of the normal distribution curve, and user10 and user21 are located at the left. Due to the presence of a small margin in data (75%), the outliers detected from talk aloud data were from user9, user10, and user21, holding the values [1, 3.5, 3.2] respectively.

As previously mentioned in the above section regarding the correctness, which reflected the mean data addressing the snippets, but now in this section, we are calculating the mean correctness percentage value of individual participants to identify outliers. To have an estimate of the correctness value, Z-score is calculated for individual participants, and plot the normal distribution curve as in Figure 5.2 shows the normal distribution of the score of the correctness percentage values. where,

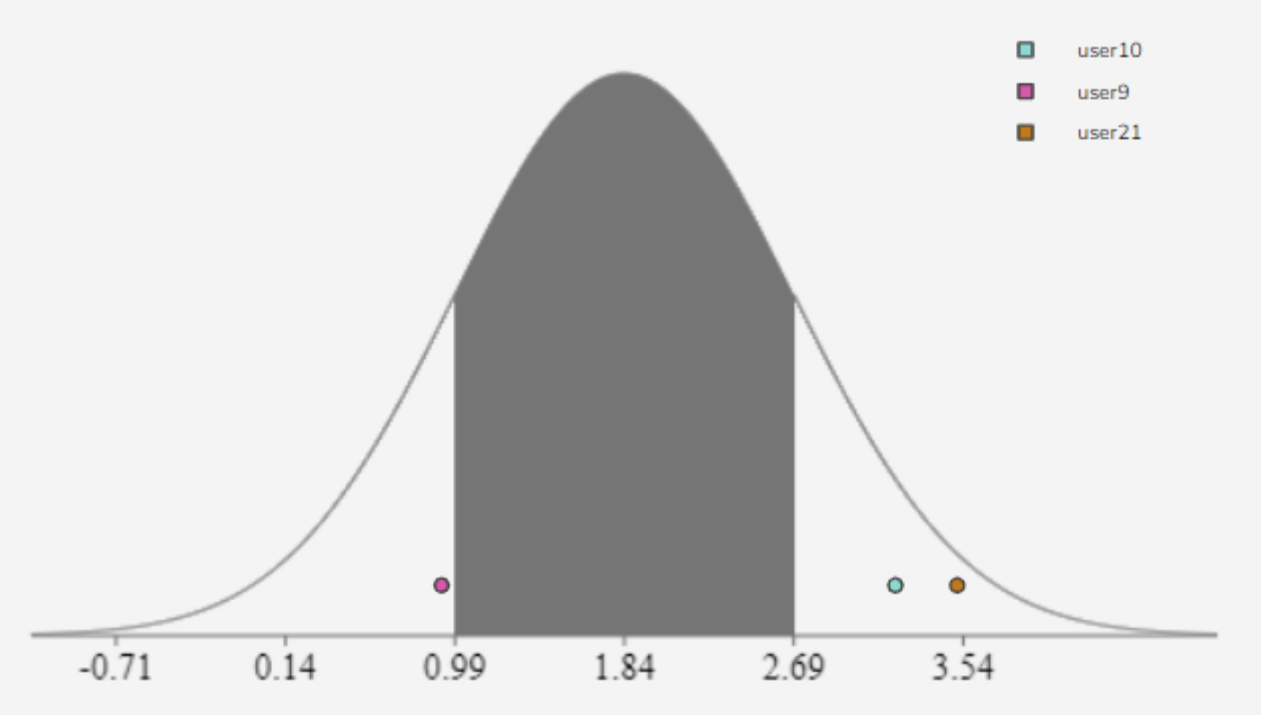

Figure 5.2: Gaussian Graph for Outlier detection from Post talk aloud data

$$Mean\ (\mu) = 5.94$$
$$Standard\ deviation\ (\delta) = 2.13$$
$$2.13 < (\mu) < 8.91$$

In this passage, considering all the values between 2.13 and 8.91, which lie within a standard deviation of ±1.7 and remain to be considered outliers. All the values of user9, user10, user20, and user21 are significantly away from the mean. User9 is located at the extreme right of the normal distribution curve, whereas users 10 and 21 are located at the left. Considering 80% of data and the mean normal distribution curve, the outlier detected from the correctness response are user9, user10, user20, and user21 with values [9, 2, 9, 2], respectively.

To maintain the data stability, User9, User10, and User21 were detected as outliers both in Correctness and Talk aloud data and were required to eliminate the data from the research and analysis part as to contribute and obtain significant statistical tests to evaluate the hypothesis.

5.2.2 Descriptive Statistics

The behavioral data differs in both group which can be observed in table 5.4 and visualized in figure 5.4 and 5.5. After removing the outliers, there were a total of 9 members in the trained group and seven members in the untrained group that

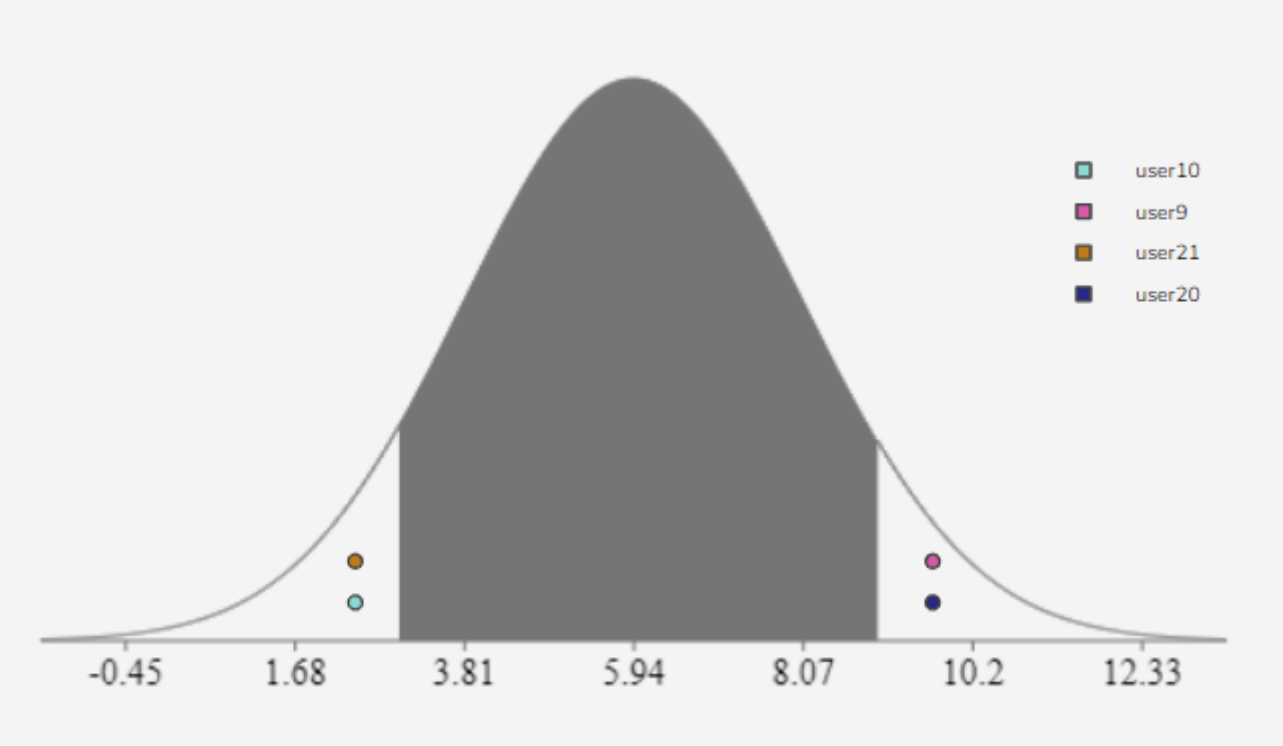

Figure 5.3: Gaussian Graph for Outlier detection from Correctness data

participated in the analysis process. Here in this section, we discuss the RQ1 for behavioral data in two parts, and response time follows the correctness. Through observation, it can be said that the response time ratio is higher in the untrained group because the trained group was initially introduced to the shortcuts prior to the experimental study, which helped them to comprehend the code faster than the untrained group.

All response time ratios were approximately bearing the same magnitude. Interestingly, data manifested reveal discernible variations in response times amongst algorithms. For example, comprehension time taken for tasks "Basic calculator," "Anagram,",and "Vowels and Consonants" shows significant differences where the time taken was less by 52.6%, 45.6% and 47.1% by the untrained group. Other tasks such as "Palindrome," "Alphabet Frequency," and "Random Card" also exposed a similar response, where the trained group took comparatively 25% less time than the untrained group.

Figure 5.5 shows the correctness data in percentage for each exercise, as one can easily observe that there was an enormous difference in some of the tasks such as the "Basic calculator" where 33.33% from the trained group and 85.71% from the untrained group provided the correct answers. In the task "Max-Min from a Matrix," 55.55% from the trained and 100% from the untrained group submitted correct answers, and in the task "Vowels and Consonants," 33.33% from the trained and 71.42% from the untrained group submitted accurate answers. On the other hand,

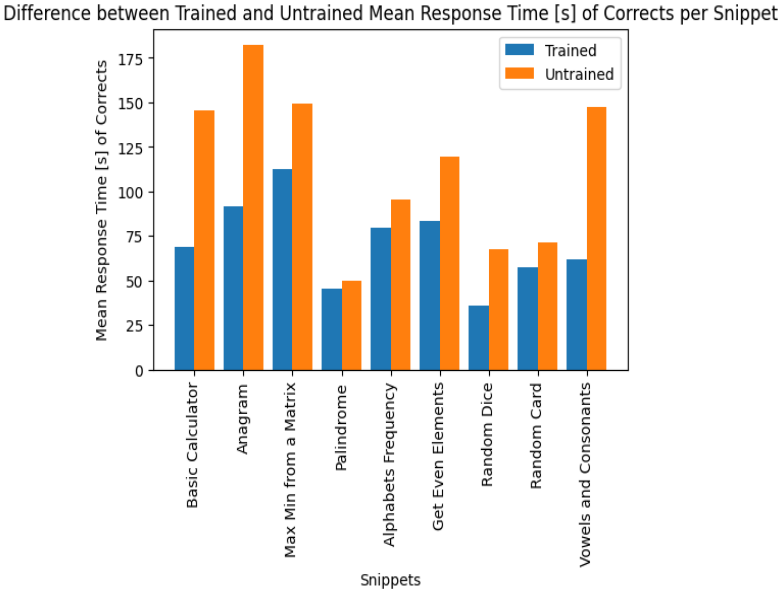

Figure 5.4: Response Time Comparison of trained and untrained group for each exercise

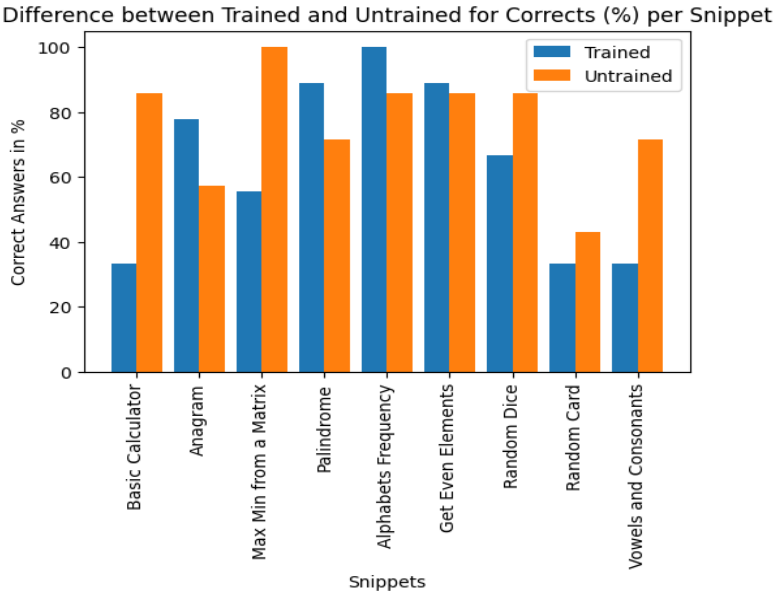

Figure 5.5: Correctness Comparison of trained and untrained group for each exercise

the trained group submitted correct answers to the task "Anagram," "Palindrome," "Alphabet Frequency," and "Get Even Elements." In the remaining tasks, there were some observable differences that can be noticed in relation to the responses given by both groups. But to summarize our observation, it can be said that the untrained group performed well as the trained group.

5.2.3 Inferential Statistics

To evaluate whether there is a significant difference in the response time values between both groups, we performed a t-test.

Shortcut Snippet	Lines of Code (LOC)	Response Time [s]	
		Trained (n1 = 9)	Untrained (n2 = 7)
Basic Calculator	22	96±54	151±47
Anagram	12	88±53	205±117
Max Min from a Matrix	22	172±142	168±80
Palindrome	7	68±53	53±25
Alphabets Frequency	14	117±91	117±49
Get Even Elements	12	111±85	139±66
Random Dice	12	60±41	90±59
Random Card	15	100±74	73±34
Vowels and Consonants	14	73±35	212±152

Table 5.2: Response Time Values of trained and untrained group for each exercise

Values	Trained	Untrained
Sample Size	n1 = 9	n2 = 7
Mean Value	μ1 = 70.67	μ2 = 114.23
Standard Deviation	σ1 = 22.44	σ2 = 42.75
P-value	0.05	
T-value	0.274	

Table 5.3: Response Time Statistical Values of trained and untrained group

The values for the hypothesis considered are:

H_0 = There is no significant difference in "Response Time" between both groups.
H_1 = There is significant difference in "Response Time" between both groups.

Statistically, we found there is a significant difference in the response time of trained and untrained groups with shortcut comprehension. The statistical data can be observed in table 5.3 where the p-value is 0.05, the sample size of the trained group is nine, and the untrained group is 7. The mean value of trained group is $\mu = 70.67$ and the standard deviation as $\delta = 22.44$ and whereas for the untrained group is $\mu = 114.23$ and the standard deviation as $\delta = 42.75$ After executing the t-test we obtained the value t = 0.274 which helped us in reaching the conclusion that Hypothesis H_1 is true.
To evaluate whether there is a significant difference in the correctness ratio, for this, we perform the same t-test.

Shortcut Snippet	Trained (n1 = 9)		Untrained (n2 = 7)	
	No of Correct answer	Mean RT [s] of Corrects	No of Correct answer	Mean RT [s] of Corrects
Basic Calculator	3	145.26	6	68.92
Anagram	7	166.54	4	91.57
Max Min from a Matrix	5	144.89	7	112.32
Palindrome	8	52.37	5	45.06
Alphabets Frequency	9	92.35	6	79.62
Get Even Elements	8	116	6	83.33
Random Dice	6	78.58	6	35
Random Card	3	69.59	5	57.17
Vowels and Consonants	3	117.14	5	62

Table 5.4: Correctness Values in [%] of trained and untrained group for each exercise

The values for the hypothesis considered are:

H_2 = There is no significant difference in "Correctness" between both groups.
H_3 = There is significant difference in "Correctness" between both groups.

We found no statistically significant difference in the correctness of trained and untrained groups with shortcut comprehension. The statistical data can be seen in table 5.5 where the p-value is 0.05, the sample size of the trained group with 9 participants, and the untrained group with 7. The mean value of trained group is $\mu = 5.78$ and the standard deviation as $\delta = 2.25$ and whereas for the untrained

Values	Trained	Untrained
Sample Size	n1 = 9	n2 = 7
Mean Value	μ1 = 5.78	μ2 = 5.33
Standard Deviation	σ1 = 2.25	σ2 = 1.15
P-value	0.05	
T-value	0.0213	

Table 5.5: Correctness Statistical Values of trained and untrained group

group is $\mu = 5.33$ and the standard deviation as $\delta = 1.15$ After executing the t-test, we obtained the value t = 0.0213, which helped us in reaching the conclusion that Hypothesis H_2 is true.

5.2.4 Conclusion

After excluding the outliers (User9, User10, and User21) data from the data set we analysed the Descriptive Statistics and Inferential Statistics on the Behavioral Data. Observing the descriptive data we can claim that the trained group had been taught the shortcuts before the experimental study, they were able to comprehend the code more quickly than the untrained group, which is why the response time ratio was higher in the trained group. In order to sum up our findings for the Correctness data, it can be claimed that the both the groups performed same. The Hypothesis clearly provided enough evidence to reach our conclusion. After the execution of statistical tests respectively for response time and correctness, the statistical results obtained provided an observation that there was a significant difference in the Response Time and no significant difference in the Correctness data.

Answers to RQ1: Overall, we found no indication in the behavioral data that shortcuts affect students' performance in program comprehension. We found significant difference in the response time but on other hand the performance compared between both groups was equal implied by correctness values.

5.3 RQ2: Visual Data

5.3.1 Pre Processing and Preparation

Correlation Analysis

A correlation study can show important connections between several measures or sets of metrics. Even though the measurements originate from many business de-

partments, information about those links might offer fresh perspectives and highlight interdependencies. Understanding which metrics have strong relationships, that is, when one metric behaves in a certain way, one or more additional metrics can be expected to behave similarly or in the opposite way; this is helpful when we track eye-tracking metrics in many different ways. The degree of change in one variable as a result of the other's change is determined via correlation analysis. We can conclude that the other variable or metric is also being impacted in a similar way if there is evidence of a high connection between the two and one of them is acting in a certain way.
A high correlation indicates a significant link between the two measurements, whereas a low correlation indicates a poor association between the two metrics. A positive correlation indicates that both measurements grow in tandem, whereas a negative correlation indicates that as one metric grows, the other one shrinks. This makes it easier to combine comparable values together and eliminates the need for separate data processing.
The figure 5.6 illustrates the correlation matrix that displays how closely two variables are correlated. It provides the correlation between each set of potential value pairings in matrix form. To summarize a vast eye tracking metric, find the patterns, and base a decision on them, we can utilize a correlation matrix. Additionally, we can display our results and determine which variable is more connected with which variable. The correlation coefficient is contained in each cell of a matrix. The coefficient values range from ± 1. Strong positive correlations are defined as values more than 0.9, and strong negative correlations as values lower than -0.9. Other than those values, there is no correlation in the relation of eye tracking metrics. The following can be observed by reading the figure 5.6 with a circle in their correlation coefficient:

1. The relation is stronger between "NoOfFixationsInAOI" and "Revisits" with a correlation coefficient of 0.94.

2. The relation is stronger between "NoOfFixationsInAOI" and "TotalDurationInAOI" with a correlation coefficient of 0.93.

3. The relation is stronger between "FirstPassDuration" and "TotalDurationOfFixationInAOI" with a correlation coefficient of 0.9.

For instance, if we examine the value in table 5.6 and the bar graphs 5.7, 5.8, we can see that when "NoOfFixationsInAOI" has a positive impact, "Revisits" likewise increases in the equal magnitude and in proportion to that value for lower values. We use "FirstPassDuration," "SecondPassDuration," and "NoOfFixationsInAOI" for our subsequent statistical analysis after correlated features have been removed.

5.3.2 Descriptive Statistics

Through Figure 5.9 the "FirstPassDuration" data can be visualized for each exercise to provide an overview of the data, followed by all other features discussed in the

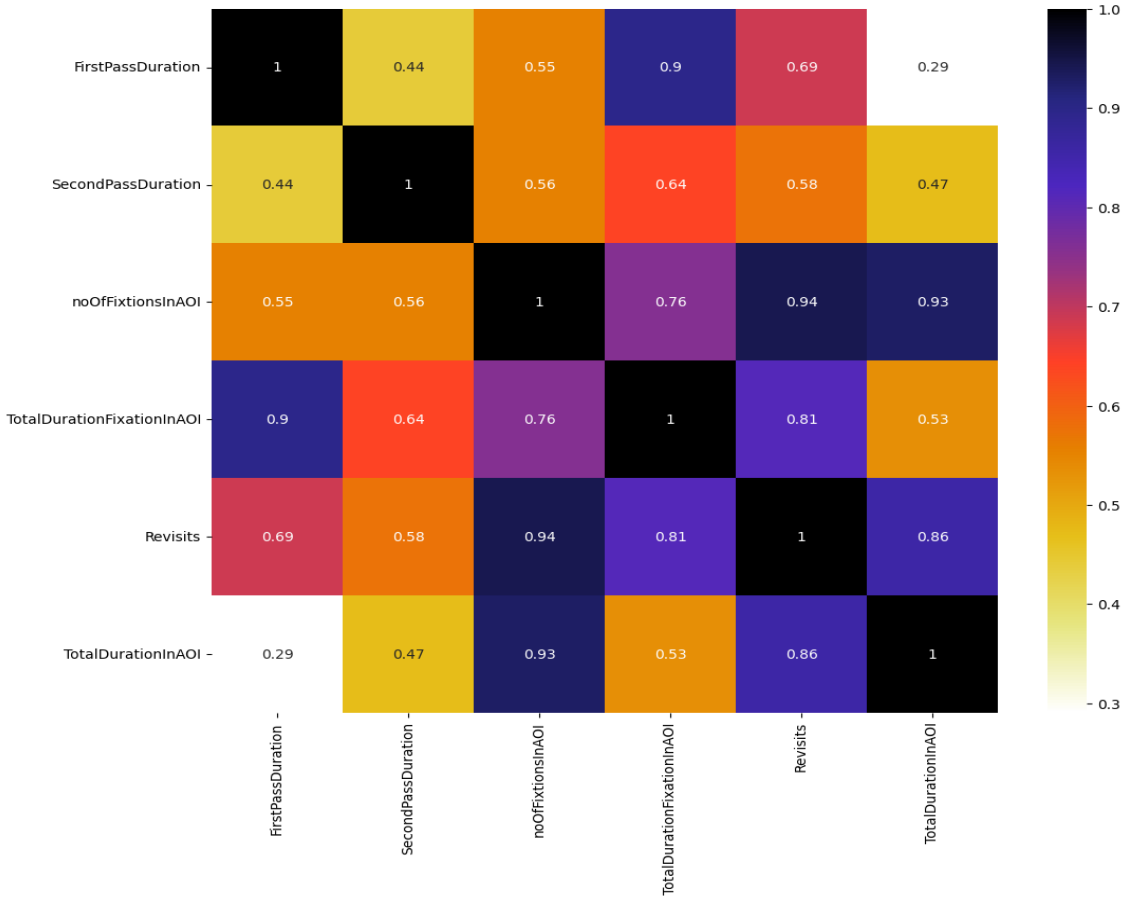

Figure 5.6: Eye tracking features values correlation comparison

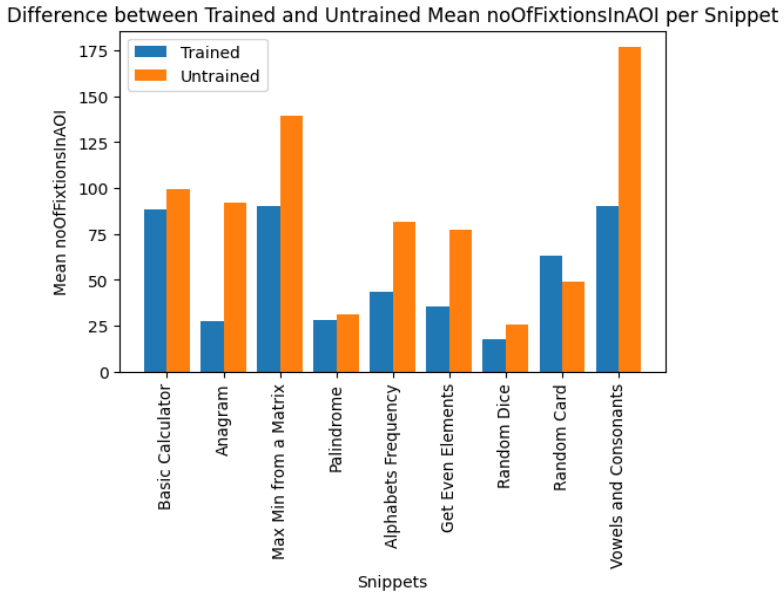

Figure 5.7: No Of Fixations in AOI Comparison of trained and untrained group for each exercise

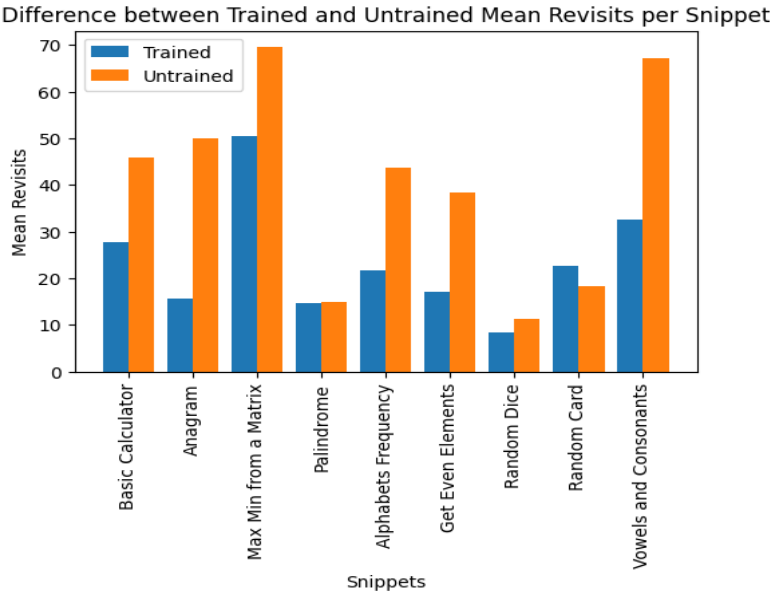

Figure 5.8: Revisits in AOI Comparison of trained and untrained group for each exercise

Shortcut Snippet	No of Fixations in AOI		Revisits	
	Trained	Untrained	Trained	Untrained
Basic Calculator	88	99.42	27.75	45.86
Anagram	27.56	92.29	15.56	50
Max Min from a Matrix	90.37	139.28	50	69.57
Palindrome	28	31.14	14.78	15
Alphabets Frequency	43.45	81.71	21.78	43.71
Get Even Elements	35.33	77.43	17.11	38.28
Random Dice	17.71	25.43	8.42	11.28
Random Card	63.38	48.86	22.62	18.29
Vowels and Consonants	90.44	176.86	32.67	67.14

Table 5.6: Number of Fixations and Revisits Values of trained and untrained group for each exercise

above section. After observing, all the exercises possess a huge difference in time spent in the first pass, where the untrained group takes five times more time than the trained group in the shortcut AOI. The figure 5.10 shows the data of "Second-PassDuration," where it is noticeable that in some of the exercises, it was similar to "FirstPassDuration" except, if we observed "SecondPassDuration" the tasks such as the Max min from a matrix, Get Even elements, random cards, and vowels and consonants the time taken in the second pass was same or more than for trained group and in the rest of the task the time taken was the same as "FirstPassDuration" approx to 5 times more.
In figure 5.7 we can see the comparison of "noOfFixtionsInAOI" for both groups. It shows that less number of fixation points in the shortcut AOIs are for the trained group than in the untrained group except for the data for the task random card where the "noOfFixtionsInAOI" are higher than those of the untrained group. As the "noOfFixtionsInAOI" and Revisits are equally correlated to each other, we can also state that the number of revisits in shortcut AOIs was lower in the trained group.

5.3.3 Inferential Statistics

To evaluate whether there is a significant difference in the "FirstPassDuration," to find out t-test was carried out. The values for the hypothesis considered are:

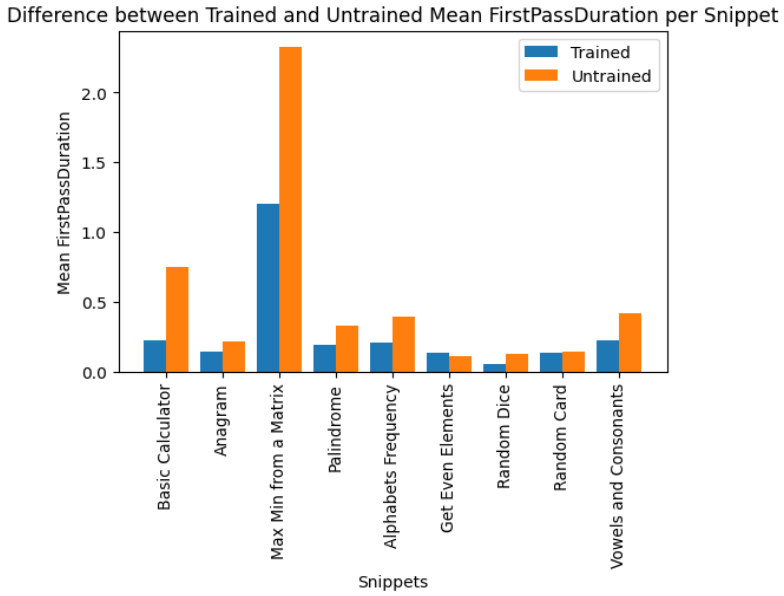

Figure 5.9: First Pass Duration in AOIs Comparison of trained and untrained group for each exercise

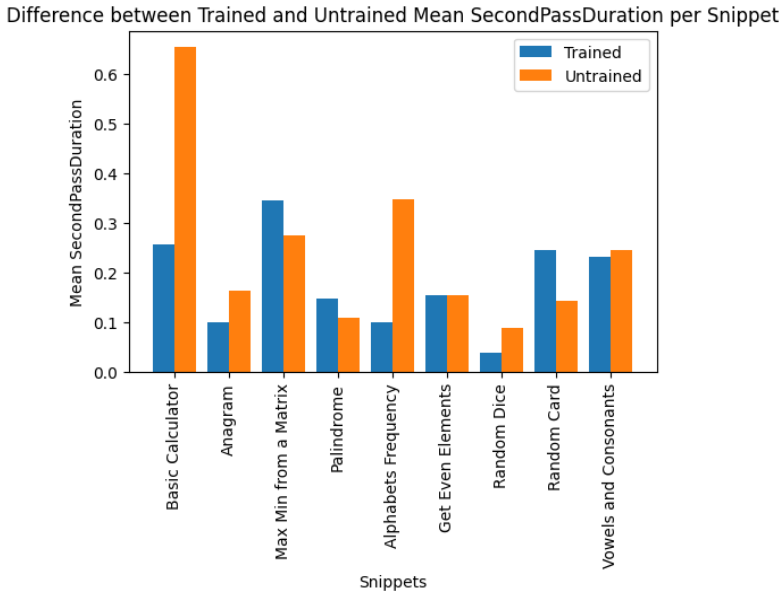

Figure 5.10: Second Pass Duration in AOIs Comparison of trained and untrained group for each exercise

Shortcut Snippet	Duration of Fixation in AOI [s]		First Pass Duration in AOI [s]		Second Pass Duration in AOI [s]	
	Trained	Untrained	Trained	Untrained	Trained	Untrained
Basic Calculator	6.65	13.87	0.23	0.75	0.26	0.65
Anagram	1.88	5.03	0.14	0.21	0.1	0.16
Max Min from a Matrix	8.33	22.18	1.2	2.33	0.34	0.27
Palindrome	1.53	1.56	0.18	0.32	0.14	0.11
Alphabets Frequency	3.72	4.99	0.2	0.39	0.1	0.35
Get Even Elements	2.01	4.8	0.13	0.11	0.15	0.15
Random Dice	0.67	1.5	0.05	0.12	0.03	0.09
Random Card	4.12	3.54	0.13	0.14	0.24	0.14
Vowels and Consonants	4.52	8.78	0.22	0.42	0.23	0.24

Table 5.7: Eye Tracking features Values of trained and untrained group for each exercise

Values	Trained	Untrained
Sample Size	n1 = 9	n2 = 7
Mean Value	μ1 = 0.275	μ2 = 0.532
Standard Deviation	σ1 = 0.331	σ2 = 0.664
P-value	0.05	
T-value	0.115	

Table 5.8: First Pass Duration Statistical Values of trained and untrained

H_4 = There is no significant difference in "FirstPassDuration" between both groups.
H_5 = There is a significant difference in "FirstPassDuration" between both groups.

We found no statistically significant difference in the "FirstPassDuration" of trained and untrained groups with shortcut comprehension. The statistical data can be observed in table 5.8 where the p-value is 0.05, the sample size of the trained group is nine, and the untrained group is 7. The mean value of trained group is $\mu = 0.275$ and the standard deviation as $\delta = 0.331$ and whereas for the untrained group is

$\mu = 0.532$ and the standard deviation as $\delta = 0.664$ After executing the t-test we obtained the value t = 0.345 which helped us in reaching the conclusion that Hypothesis H_5 is true.

Values	Trained	Untrained
Sample Size	n1 = 9	n2 = 7
Mean Value	μ1 = 0.177	μ2 = 0.24
Standard Deviation	σ1 = 0.092	σ2 = 0.165
P-value	0.05	
T-value	0.36	

Table 5.9: Second Pass Duration Statistical Values of trained and untrained

To evaluate whether there is a significant difference in the "SecondPassDuration," to find out, we performed a t-test. The values for the hypothesis are considered here:

H_6 = There is no significant difference in "SecondPassDuration" between both groups.
H_7 = There is a significant difference in "SecondPassDuration" between both groups.

We found no statistically significant difference in the "SecondPassDuration" of trained and untrained groups with shortcut comprehension. The statistical data can be observed in table 5.9 where the p-value is 0.05, the sample size of the trained group is nine, and the untrained group is 7. The mean value of trained group is $\mu = 0.177$ and the standard deviation as $\delta = 0.092$ and whereas for the untrained group is $\mu = 0.24$ and the standard deviation as $\delta = 0.165$ After executing the t-test we obtained the value t = 0.36 which helped us in reaching the conclusion that Hypothesis H_7 is true.

To evaluate whether there is a significant difference in the "noOfFixtionsInAOI," to find out, we performed a t-test. The values for the hypothesis considered are:

H_8 = There is no significant difference in "noOfFixtionsInAOI" between both groups.
H_9 = There is a significant difference in "noOfFixtionsInAOI" between both groups.

We found no statistically significant difference in the "noOfFixtionsInAOI" of trained

Values	Trained	Untrained
Sample Size	n1 = 9	n2 = 7
Mean Value	μ1 = 50.8	μ2 = 85.82
Standard Deviation	σ1 = 27.98	σ2 = 46.5
P-value	0.05	
T-value	0.345	

Table 5.10: Number of Fixtions In AOI Statistical Values of trained and untrained

and untrained groups with shortcut comprehension. The statistical data can be observed in table 5.10 where the p-value is 0.05, the sample size of the trained group is nine, and the untrained group is 7. The mean value of trained group is $\mu = 50.8$ and the standard deviation as $\delta = 27.98$ and whereas for the untrained group is $\mu = 85.82$ and the standard deviation as $\delta = 46.5$. After executing the t-test we obtained the value t = 0.115 which helped us in reaching the conclusion that Hypothesis H_9 is true.

5.3.4 Conclusion

After correlation analysis, we observed that the when "NoOfFixationsInAOI" has a positive impact, "Revisits" likewise increases in the equal magnitude and in proportion to that value for lower values. We exclude the "Revisits" for statistical tests and carry our the Descriptive Statistics and Inferential Statistics on the Visual Data. The "FirstPassDuration" and "SecondPassDuration" in the AOI for all exercises vary enormously, with the untrained group taking five times longer than the trained group. It demonstrates that the trained group had fewer fixation points in the shortcut AOIs than the untrained group. The Hypothesis clearly provided enough evidence to reach our conclusion. After the execution of statistical tests it states that there is significant difference in all the eye tracking metrics. The results obtained provided a structure of observation that the training offered to the students prior to the test had a significant difference in the Visual data in comprehending the shortcuts. It is adequate to state that there is a significant difference in visual attention in both groups.

Answers to RQ2: Overall, we found indication in the visual data that shortcuts affect students' performance in program comprehension. There was significant difference in Visual data for all the eye tracking metrics.

6 Discussion

During the data analysis, we discovered a number of intriguing findings, which we now go on to discuss. We explicitly distinguish between solving the research questions and data exploration that is not covered by our research questions. If we observe patterns in the behavioral data, we can conclude that the untrained group performed well in answering the question than the trained group can be the case because the function names gave them a clue what is the role of the function itself while the trained group tried to comprehend the statements precisely and lost their way through it. We can also see the difference in response time in the trained group was lower because of the introduction of shortcut statements prior to the test, and the untrained group took more time because the focus point was more on the shortcut code, which was the challenging part to grasp.
The basic approach and computations are the focus of the majority of the programs, but we also discovered that participants' visual attention is drawn to programs with the introduction of the pretest. We examined and retrieved pertinent statistics from the visual-attention data the Tobii Pro Fusion eye tracker had generated. Heatmaps, fixation diagrams, and scan paths were constructed by taking all the relevant data into account. To get assistance in Inferential Statistics and test the hypothesis, we also extracted eye tracking feature values in the area of interest (AOIs).
AOIs are defined in accordance with their involvement in the code snippet, which includes :

1. 'Declaration and/or usage of shortcut' The coordinates of the code segment where the actual shortcut code lies and is used for extracting feature values.

2. 'Statements that call the shortcut' The code fragment that calls the shortcut code or the function where the shortcut code has been used.

3. 'Return Result' - The final print statement code for returning the result.

Figure 6.1,6.3 and 6.4, the diagram represents the heatmap, fixation diagrams, and scan path with the AOIs. This adds an explanation to the degree to which the participant gazed in AOIs. Differences between the trained group and the untrained group can be observed, where the heatmap and fixation points are spread across the code snippet. In addition, with the focus shift between both groups in the scan path diagram. We found the evident difference between both groups, but our primary concern was how does the participant perceive the shortcut code and other important code fragments. Another difference can be observed in the fixation diagram and scan

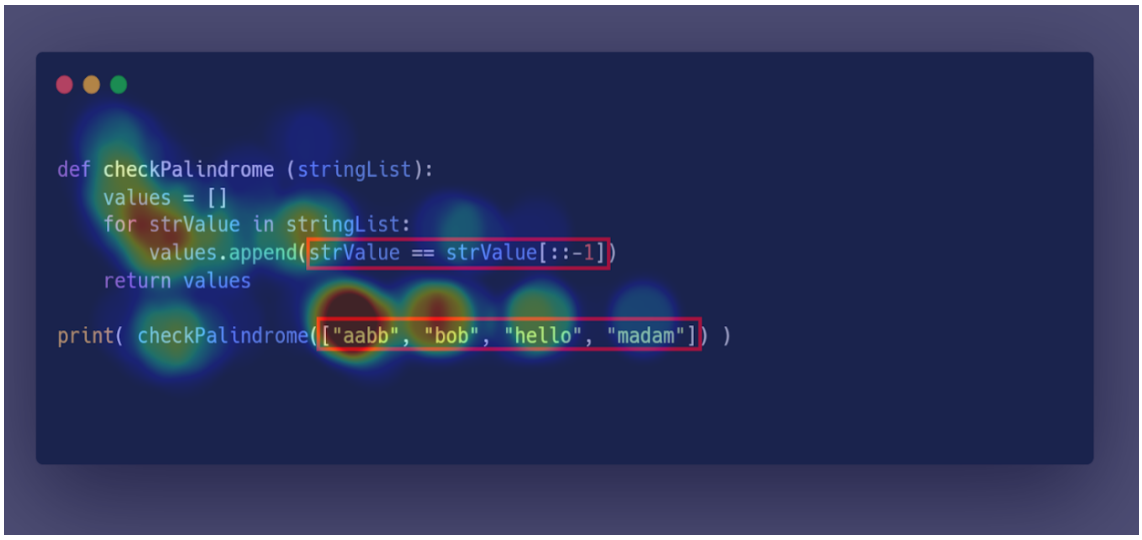

Figure 6.1: Palindrome exercise with correct response

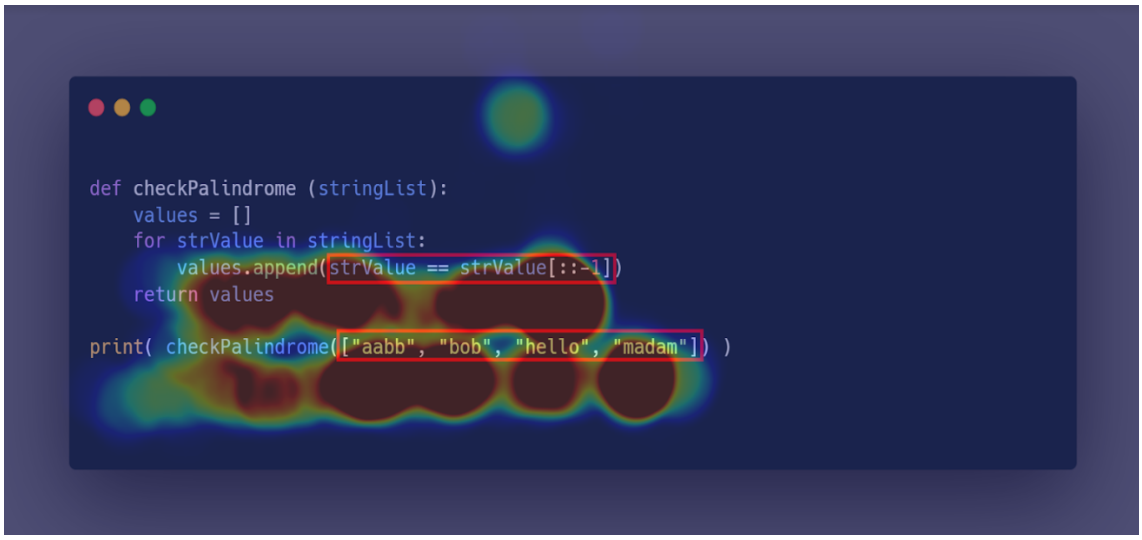

Figure 6.2: Palindrome exercise with incorrect response

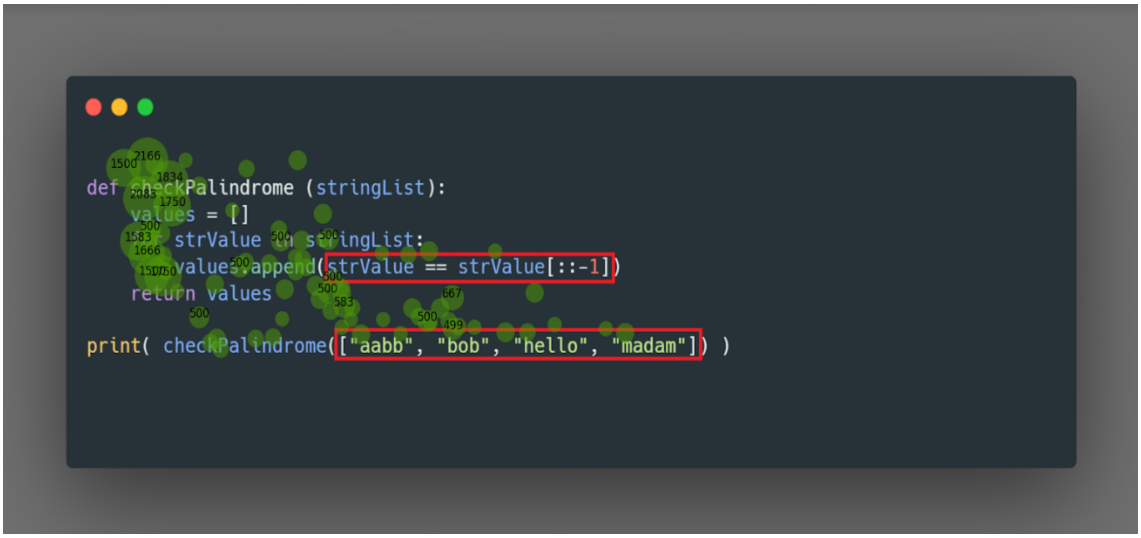

Figure 6.3: Fixation Points of Palindrome Exercise for correct response

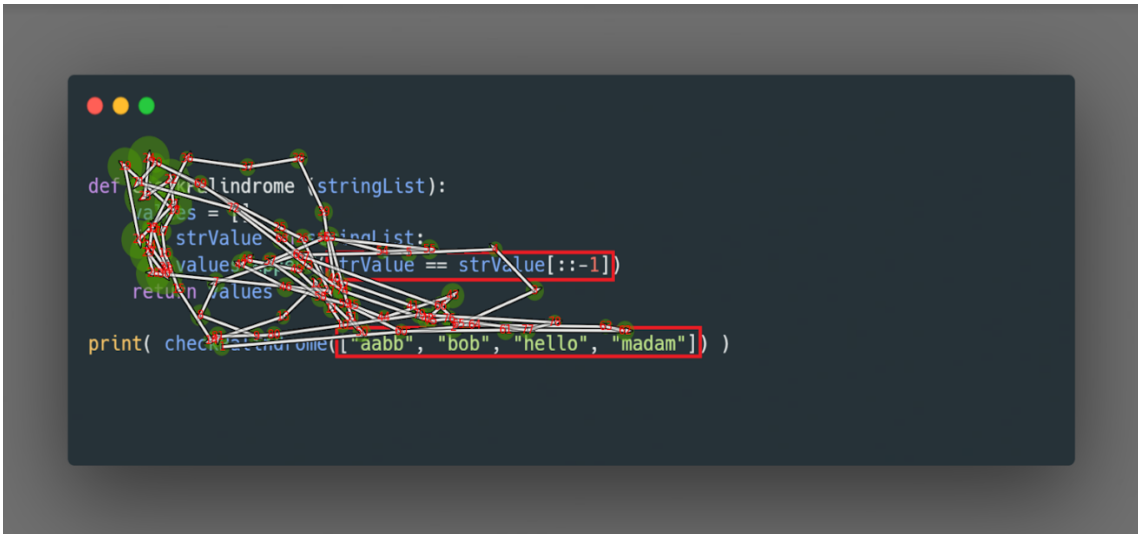

Figure 6.4: Scan Path of Palindrome Exercise for correct response

path by looking at the number of fixation points and their duration and the number of times the participant revisited the AOIs in the flow of comprehension.

After observing the visual diagrams, the code fragment with the shortcut seen by the untrained group gave a lot of weight to it.

7 Threat to Validity

7.1 Internal Threat

Our study and the participant sample pose a number of risks to validity. First, the gender distribution is angled, but it still closely mirrors the demography in computer science at our institutes. Second, we have to wonder if the classification of participants into groups was valid: Do our intermediate programmers have enough programming experience? Based on a questionnaire, we asked participants a few questions about their experiences to ensure a proper assignment.
Although controlled eye tracking studies must account for confounding variables to the greatest extent possible, we deliberately focused on strong internal validity. We chose a homogeneous sample of programmers with a range of programming expertise (trained group), and our snippets are rather brief and written in just one programming language. As a result, our findings are limited to identical situations and cannot be easily extended to other situations, such as experienced programmers or vast code bases. There must be a trade-off between aiming for high internal validity and high exterior validity.
It's important to note that although our experiment focused on methods, software systems actually contain a variety of techniques and higher-level components. Our findings nevertheless have relevance in real-world situations: Code complexity measures might help predict cognitive effort and that they might demand more time than anticipated for intermediate programmers when we know that they work at the method level. Future research will cover other complexity metrics that have been developed above the technique level. Our research serves as a springboard for specialized follow-up investigations into these measurements and related cognitive processes.

7.2 External Threat

Our study demonstrates the typical dangers of using student enrollment and small Python scripts. Only a very careful translation of our results to additional contextual elements is possible. Reading behavior of longer snippets with more sophisticated control flows may produce various outcomes. It's important to note that although our experiment focused on methods, software systems actually contain a variety of techniques and higher-level components. Our findings nevertheless have relevance in real-world situations: Code complexity measures might help predict cognitive effort and that they might demand more time than anticipated for intermediate

programmers when we know that they work at the method level. Future research will cover other complexity metrics that have been developed above the technique level. Our research serves as a springboard for specialized follow-up investigations into these measurements and related cognitive processes.

8 Conclusion and Future Work

The findings showed that participants took less time to understand the shortcuts after receiving training before the test. Even though the correctness response was were similar in both group the time taken to understand the code was significant less by trained group. The trained group gazed through the key statements in the code snippets and followed the execution flow. If we observe the number of revisit and fixations in trained group it states that they understood the shortcut code statement with less repetition of gaze activity. The focus point was more in area of interest than in non important code statements. Training the student add much more value in understand the code with shortcut statement(s).

If we observe the scan path with AOIs, we can observe that did the participant seen the execution flow of the shortcut? Did they noticed the role of the shortcuts played in the code? Did they gazed through the key elements in the code snippet? Were the participant was confused by the non-important code fragment outside the AOIs? Using the visual diagrams' noted focus areas as a guide, we retrieved the following patterns. Observing the scan path with AOIs provokes follow-up questions:

1. Did the participant see the execution flow of the shortcut?
2. Did they notice what the role of shortcuts is?
3. Did they gaze through the key elements in the code snippet?
4. Were the participant confused by the non-important code fragment outside the AOIs?

If we look at the non-important code fragment with the same level of heat, we can postulate from this that the participant was gazing through the whole code, or it brings us to another question, whether the student was getting distracted because the heatmap was spread all over the snippet.
Our results demonstrate that the introducing shortcuts approach has visible influence on how well students comprehend programs with shortcuts. We also emphasize the need for future research. We were unable to measure cognitive load (e.g., with pupil dilation), see how students processed bits of information, or use more complex snippets because of the extremely constrained nature of our environment. The students can be instructed to code their own program utilizing shortcuts. Instead of using post talk aloud session a during talk aloud session will help in answers question like how, when and why the students gazed through certain part of code.

Bibliography

[1] A 5 minute guide to numba — numba 0.46.0.dev0+566 ... - pydata, 2012. (Online Accessed: 09.08.2022).

[2] Overview — numba 0.17.0-py2.7-linux-x86_64.egg documentation, 2012. (Online Accessed: 09.08.2022).

[3] Eye tracking for pupillometry, 2020. (Online Accessed: 10.08.2022).

[4] What is eye tracking?, December 2020. (Online Accessed: 10.08.2022).

[5] 10 most used eye tracking metrics and terms - imotions., 2021.

[6] Motivation - cuda python 11.7.1 documentation - github pages, 2021. (Online, Accessed: 08.08.2022).

[7] Overview - cuda python 11.7.1 documentation - github pages, 2021. (Online, Accessed: 08.08.2022).

[8] How does blinking affect eye tracking?, 2022. (Online Accessed: 10.08.2022).

[9] David Adamo. An experiment to measure the cognitive weights of code control structures, November 2015.

[10] Shulamyt Ajami, Yonatan Woodbridge, and Dror G. Feitelson. Syntax, predicates, idioms - what really affects code complexity? In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 66–76, 2017.

[11] Gennady Andrienko, Natalia Andrienko, Michael Burch, and Daniel Weiskopf. Visual analytics methodology for eye movement studies. *IEEE transactions on Visualization and Computer Graphics*, 18(12):2889–2898, 2012.

[12] Shlomo Berkovsky, Ronnie Taib, Irena Koprinska, Eileen Wang, Yucheng Zeng, Jingjie Li, and Sabina Kleitman. Detecting personality traits using eye-tracking data. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[13] Dave Binkley, Marcia Davis, Dawn Lawrie, and Christopher Morrell. To camelcase or under$_s core.pages 158 - -167, 2009$.

[14] Ruven Brooks. Towards a theory of the comprehension of computer programs. (543-554), 1983.

[15] Ruven Brooks. Towards a theory of the comprehension of computer programs. *International journal of man-machine studies*, 18(6):543–554, 1983.

[16] Lisa Bruttel and Irenaeus Wolff. Incentives and random answers in post-experimental questionnaires. (110), 2018.

[17] Raluca Budiu. Between-subjects vs. within-subjects study design, May 2018. (Online Accessed: 15.08.2022).

[18] Jean-Marie Burkhardt, Françoise Détienne, and Susan Wiedenbeck. Object-oriented program comprehension: Effect of expertise, task and phase. *Empirical Software Engineering*, 7(2):115–156, 2002.

[19] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. Eye movements in code reading: Relaxing the linear order. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 255–265. IEEE, 2015.

[20] Nelson Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and brain sciences*, 24(1):87–114, 2001.

[21] Kirsten A Dalrymple, Ming Jiang, Qi Zhao, and Jed T Elison. Machine learning accurately classifies age of toddlers based on eye tracking. *Scientific reports*, 9(1):1–10, 2019.

[22] Andrew T Duchowski and Andrew T Duchowski. *Eye tracking methodology: Theory and practice*. Springer, 2017.

[23] Rodrigo Duran, Juha Sorva, and Sofia Leite. Towards an analysis of program complexity from a cognitive perspective. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 21–30, 2018.

[24] O Isola Esther, O Olabiyisi Stephen, O Omidiora Elijah, A Ganiyu Rafiu, T Ogunbiyi Dimple, and Y Adebayo Olajide. Development of an improved cognitive complexity metrics for object-oriented codes. *Br. J. Math. Comput. Sci*, 18(2):1–11, 2016.

[25] Bryn Farnsworth. Eye tracking: The complete pocket guide, August 2022. (Online Accessed: 12.08.2022).

[26] Kathi Fisler. The recurring rainfall problem. In *Proceedings of the tenth annual conference on International computing education research*, pages 35–42, 2014.

[27] Kathi Fisler, Shriram Krishnamurthi, and Janet Siegmund. Modernizing plan-composition studies. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 211–216, 2016.

[28] Richard K Fjeldstad. Application program maintenance study. *Report to Our Respondents, Proceedings GUIDE*, 48, 1983.

[29] Robert W Floyd. The syntax of programming languages-a survey. *IEEE Transactions on Electronic Computers*, (4):346–353, 1964.

[30] Judith Gal-Ezer and David Harel. What (else) should cs educators know? *Communications of the ACM*, 41(9):77–84, 1998.

[31] Kerstin Gidlöf, Nils Holmberg, and Helena Sandberg. The use of eye-tracking and retrospective interviews to study teenagers' exposure to online advertising. *Visual Communication*, 11(3):329–345, 2012.

[32] Stephanie Glen. Correlation coefficient: Simple definition, formula, easy steps, 2022. (Online Accessed: 18.08.2022).

[33] Richard Gluga, Judy Kay, Raymond Lister, and Donna Teague. On the reliability of classifying programming tasks using a neo-piagetian theory of cognitive development. In *Proceedings of the ninth annual international conference on International computing education research*, pages 31–38, 2012.

[34] Volker Gruhn and Ralf Laue. On experiments for measuring cognitive weights for software control structures. In *6th IEEE International Conference on Cognitive Informatics*, pages 116–119, 2007.

[35] KV Hanford and Cliff B Jones. Dynamic syntax: A concept for the definition of the syntax of programming languages. *Annual review in automatic programming*, 7:115–142, 1973.

[36] ADAM HAYES. T-test: What it is with multiple formulas and when to use them, July 2022. (Online Accessed: 18.08.2022).

[37] Jan Heering and Paul Klint. Semantics of programming languages: A tool-oriented approach. *ACM Sigplan Notices*, 35(3):39–48, 2000.

[38] Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen Kraemer, Violetta Lonati, Claudio Mirolo, et al. Fostering program comprehension in novice programmers-learning activities and learning trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, pages 27–52. 2019.

[39] Leena Jain and Satinderjit Singh. Designing the code snippets for experiments on code comprehension of different software constructs. *International Journal of Computer Sciences and Engineering*, 7:310–318, March 2019.

[40] Amit Jakhar and Kumar Rajnish. A new cognitive approach to measure the complexity of software's. *International Journal of Software Engineering and its Applications*, 8:185–198, January 2014.

[41] Randall W. Engle Jason S. Tsukahara, Alexander P. Burgoyne. Pupil size is a marker of intelligence - scientific american, June 2021. (Online Accessed: 10.08.2022).

[42] Amy J Ko, Brad A Myers, Michael J Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on software engineering*, 32(12):971–987, 2006.

[43] Timo Kootstra, Jonas Teuwen, Jeroen Goudsmit, Tanja Nijboer, Michael Dodd, and Stefan Van der Stigchel. Machine learning-based classification of viewing behavior using a wide range of statistical oculomotor features. *Journal of vision*, 20(9):1–1, 2020.

[44] John R Koza, Forrest H Bennett, David Andre, and Martin A Keane. Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In *Artificial intelligence in design'96*, pages 151–170. Springer, 1996.

[45] Clifton Kussmaul. Process oriented guided inquiry learning (pogil) for computer science. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 373–378, 2012.

[46] Thomas D LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*, pages 492–501, 2006.

[47] Stanley Letovsky. Cognitive processes in program comprehension. *Journal of Systems and software*, 7(4):325–339, 1987.

[48] Mountstephens J Lim JZ and Teo J. Eye-tracking feature extraction for biometric machine learning. front. neurorobot, 2022. (Online Accessed: 09.08.2022).

[49] Raymond Lister. After the gold rush: toward sustainable scholarship in computing. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 3–17. Citeseer, 2008.

[50] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the fourth international workshop on computing education research*, pages 101–112, 2008.

[51] Alun Lucas. What eye tracking can teach us about form optimization and design, May 2021. (Online Accessed: 08.08.2022).

[52] Graham Markall. The life of a numba kernel: A compilation pipeline taking user defined functions in python to cuda kernels, June 2020. (Online Accessed: 09.08.2022).

[53] T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976.

[54] Mark A. Mento. Different kinds of eye tracking devices, June 2020. (Online Accessed: 10.08.2022).

[55] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[56] Sanjay Misra, Adewole Adewumi, Robertas Damaševičius, and Rytis Maskeliunas. Analysis of existing software cognitive complexity measures. *International Journal of Secure Software Engineering*, 8:51–71, October 2017.

[57] Greg L Nelson, Benjamin Xie, and Amy J Ko. Comprehension first: evaluating a novel pedagogy and tutoring system for program tracing in cs1. In *Proceedings of the 2017 ACM conference on international computing education research*, pages 2–11, 2017.

[58] Matthew Nicely and Keith Kraus. Unifying the cuda python ecosystem — nvidia technical blog., April 2021. (Online, Accessed: 08.08.2022).

[59] Nils J Nilsson. Learning machines. 1965.

[60] Patrick Peachock, Nicholas Iovino, and Bonita Sharif. Investigating eye movements in natural language and c++ source code-a replication experiment. In *International conference on augmented cognition*, pages 206–218. Springer, 2017.

[61] Norman Peitek, Janet Siegmund, and Sven Apel. What drives the reading order of programmers? an eye tracking study. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 342–353, 2020.

[62] Nancy Pennington. Comprehension strategies in programming. In *Empirical Studies of Programmers: Second Workshop, 1987*, pages 100–113, 1987.

[63] Nancy Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. (295-341), 1987.

[64] Roy M Pritchard. Stabilized images on the retina. *Scientific American*, 204(6):72–79, 1961.

[65] Keith Rayner. The 35th sir frederick bartlett lecture: Eye movements and attention in reading, scene perception, and visual search. *Quarterly journal of experimental psychology*, 62(8):1457–1506, 2009.

[66] Christophe Rigaud, Nam Le, Jean-Christophe Burie, Jean-Marc Ogier, Shoya Ishimaru, Motoi Iwata, and Koichi Kise. Semi-automatic text and graphics extraction of manga using eye tracking information. pages 120–125, April 2016.

[67] Robert S Rist. Schema creation in programming. *Cognitive Science*, 13(3):389–414, 1989.

[68] Martin P Robillard, Wesley Coelho, and Gail C Murphy. How effective developers investigate source code: An exploratory study. *IEEE Transactions on software engineering*, 30(12):889–903, 2004.

[69] David A Robinson. A method of measuring eye movemnent using a scieral search coil in a magnetic field. *IEEE Transactions on bio-medical electronics*, 10(4):137–145, 1963.

[70] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. How do professional developers comprehend software? In *2012 34th International Conference on Software Engineering (ICSE)*, pages 255–265. IEEE, 2012.

[71] Anna Rogalska, Filip Rynkiewicz, Marcin Daszuta, Krzysztof Guzek, and Piotr Napieralski. Blinking extraction in eye gaze system for stereoscopy movies. *Open Physics*, 17:512–518, September 2019.

[72] Nanda NJ Rommelse, Stefan Van der Stigchel, and Joseph A Sergeant. A review on eye movement studies in childhood and adolescent psychiatry. *Brain and cognition*, 68(3):391–414, 2008.

[73] DE Rumelhart. Schemata: The building blocks of cognition., chapter 2, 1980.

[74] Wolfgang Schnotz and Christian Kürschner. A reconsideration of cognitive load theory. *Educational psychology review*, 19(4):469–508, 2007.

[75] Carsten Schulte. Block model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the fourth international workshop on computing education research*, pages 149–160, 2008.

[76] Carsten Schulte, Tony Clear, Ahmad Taherkhani, Teresa Busjahn, and James H Paterson. An introduction to program comprehension for computer science educators. *Proceedings of the 2010 ITiCSE working group reports*, pages 65–86, 2010.

[77] Sue Sentance and Jane Waite. Primm: Exploring pedagogical approaches for teaching text-based programming in school. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, pages 113–114, 2017.

[78] Jingqiu Shao and Yingxu Wang. A new measure of software complexity based on cognitive weights. *Electrical and Computer Engineering, Canadian Journal of*, 28(2):69 – 74, May 2003.

[79] Bonita Sharif and Jonathan I Maletic. An eye tracking study on camelcase and under_score identifier styles. In *2010 IEEE 18th International Conference on Program Comprehension*, pages 196–205. IEEE, 2010.

[80] Ben Shneiderman and Richard Mayer. Syntactic/semantic interactions in programmer behavior: A model and experimental results. (219-238), 1979.

[81] Hao Li Siddharth Misra and Jiabo He. Machine learning for subsurface characterization. In *Machine Learning for Subsurface Characterization*, pages 1–37. ScienceDirect, 2020.

[82] Janet Siegmund. Program comprehension: Past, present, and future. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 13–20. IEEE, 2016.

[83] Mark Silberstein, Assaf Schuster, Dan Geiger, Anjul Patney, and John Owens. Efficient computation of sum-products on gpus through software-managed cache. pages 309–318, January 2008.

[84] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. An examination of software engineering work practices. In *CASCON First Decade High Impact Papers*, pages 174–188. 2010.

[85] Elliot Soloway. Learning to program= learning to construct mechanisms and explanations. *Communications of the ACM*, 29(9):850–858, 1986.

[86] Elliot Soloway and Kate Ehrlich. Empirical studies of programming knowledge. (595-609), 1984.

[87] Elliot Soloway and Kate Ehrlich. Empirical studies of programming knowledge. *IEEE Transactions on software engineering*, (5):595–609, 1984.

[88] Dionisije Sopic, Amir Aminifar, and David Atienza. e-glass: A wearable system for real-time detection of epileptic seizures. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[89] James C Spohrer and Elliot Soloway. Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, 29(7):624–632, 1986.

[90] Rich Staats. Between-subjects vs. within-subjects studies - secret stache, 2022. (Online Accessed: 12.08.2022).

[91] Rich Staats. Saccade, saccadic eye movement, 2022. (Online Accessed: 12.08.2022).

[92] John Sweller. Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational psychology review*, 22(2):123–138, 2010.

[93] Donna Teague. *Neo-Piagetian Theory and the novice programmer*. PhD thesis, Queensland University of Technology, 2015.

[94] Donna Teague and Raymond Lister. Programming: reading, writing and reversing. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 285–290, 2014.

[95] Rupert Thomas. Accelerate computation with pycuda, November 2021. (Online Accessed: 09.08.2022).

[96] Gina Venolia Thomas LaToza and Robert DeLine. Maintaining mental models: A study of developer work habits. in proc. int. conf. software engineering (icse). acm. (492-501), 2006.

[97] Rebecca Tiarks. What programmers really do: An observational study. (36-37), 2011.

[98] Franklyn Turbak and David Gifford. *Design concepts in programming languages*. MIT press, 2008.

[99] Rachel Turner, Michael Falcone, Bonita Sharif, and Alina Lazar. An eye-tracking study assessing the comprehension of c++ and python source code. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 231–234, 2014.

[100] Habib Ullah, Muhammad Uzair, Arif Mahmood, Mohib Ullah, Sultan Daud Khan, and Faouzi Alaya Cheikh. Internal emotion classification using eeg signal with sparse discriminative ensemble. *IEEE Access*, 7:40144–40153, 2019.

[101] Y Wang and S Patel. Ijssci-1201-cogfundse.pdf. *Int. J. Softw. Sci. Comput. Intell.*, 1(4):1–19, June 2009.

[102] Yingxu Wang. Cognitive complexity of software and its measurement. In *2006 5th IEEE International Conference on Cognitive Informatics*, volume 1, pages 226–235, 2006.

[103] John Wrenn and Shriram Krishnamurthi. Will students write tests early without coercion?. In *Koli Calling'20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, pages 1–5, 2020.

[104] Jingxiang Yang, Yongqiang Zhao, Jonathan Cheung-Wai Chan, and Chen Yi. Hyperspectral image classification using two-channel deep convolutional neural network. In *2016 IEEE international geoscience and remote sensing symposium (IGARSS)*, pages 5079–5082. IEEE, 2016.

[105] Johannes Zagermann, Ulrike Pfeil, and Harald Reiterer. Measuring cognitive load using eye tracking technology in visual computing. In *Proceedings of the sixth workshop on beyond time and errors on novel evaluation methods for visualization*, pages 78–85, 2016.