TECHNISCHE UNIVERSITÄT
CHEMNITZ

Eye tracking study to investigate the effect of curly brackets in program comprehension

Master Thesis

Department of Computer Science
Professorship of Web Engineering
Chemnitz University of Technology

Submitted by
Vipul Semwal
Matrikel Nr.:

Professor : Dr. Janet Siegmund

Supervisor : MSc. Arooba Aqeel

Chemnitz University of Technology

Submission date: 30.09.2022

Abstract

Program comprehension is the process through which a programmer understands the source code. There have been my theories and methodologies to better program comprehension, including as Eye-Tracking, Electroencephalogram (EEG), and Functional Magnetic Resonance Imaging(fMRI) [7][8][9]. After reviewing studies in the subject of program understanding [1][2], Eye-Tracking seems to be one of the most practical and successful methods. In earlier research, the impact of curly brackets was never examined. However, the significance of curly brackets cannot be ignored since they are an integral feature of many programming languages like C, C++, Java, Rust, etc. We performed eye tracking research in this thesis to explore the impact of several styles of curly brackets (WOC, WIC, and CNL) on program understanding. This research examined the dependent variables response time, correctness, and visual attention. The research was carried out in the laboratory of Chemnitz University of Technology, where an Eye-Tracker and PsychoPy were set up together. 20 participants took part in the study, and they were required to comprehend different code snippets. Then, after each snippet, participants were asked to choose an answer from two possibilities using a mouse click, and each participant's response was registered. In addition, an Eye-Tracker monitors the participants' eye movements throughout the study.

Response time, accuracy of responses, and visual attention have all been calculated using survey data. In our investigation, we discovered that there is no difference in response time, correctness, or visual attention when different styles of curly brackets are used. Even though we discovered some intriguing outcomes for visual attention, such as the fact that lengthier snippets in which curly brackets were started from a new line (CNL) had the highest average fixation duration.

Acknowledgments

Without the insightful direction and helpful support of many people, I would not have been able to successfully complete this work. These individuals have my greatest thanks for their contributions. I would want to express my gratitude to my supervisor **Arooba Aqeel** and my professor **Prof. Dr. Janet Siegmund** for providing me with this opportunity and for their assistance and guidance during the process of doing research for my thesis. In addition, I'd want to express my gratitude to Chemnitz University of Technology for providing the resources available. In conclusion, I would want to express my gratitude to everyone who participated in the research and offered insightful responses.

Table of Contents

List of Figures

List of Tables

1 Introduction

Program comprehension or code comprehension refers to the process by which developers interpret source code; it is an essential human factor in software engineering. Software developers’ fundamental task is to understand the program. Understanding source code is a crucial cognitive activity in software development, since developers spend most of their time doing so [1].

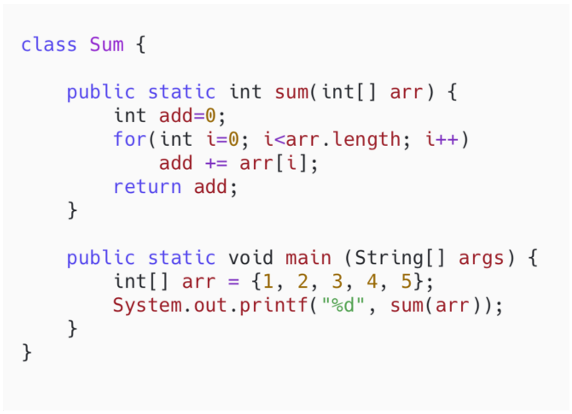
If we had a greater grasp of the cognitive process that is involved in programming, we could determine better methods to teach programming and understand how different teaching approaches would influence the programmers' ability to comprehend the programs they are working with. It helps create and build efficient programming languages and software tools that assist developers in their daily lives and help them write better software [4].

As program comprehension has been studied for a long time and in starting phase researcher used approaches like memorization, think aloud protocol. Later in time techniques like neuro imaging and Eye–Tracking started gaining popularity in program comprehension. In some studies fMRI and eye tracking are combined to get more gist about the programming comprehension [2][3]. Other studies have examined programme comprehension using near-infrared spectroscopy and electroencephalography (EEG) which is more cognitive and gives information about neuron level. In this study we are going to conduct an eye tracking study to understand the program comprehension more precisely.

1.1 Problem statement and Motivation

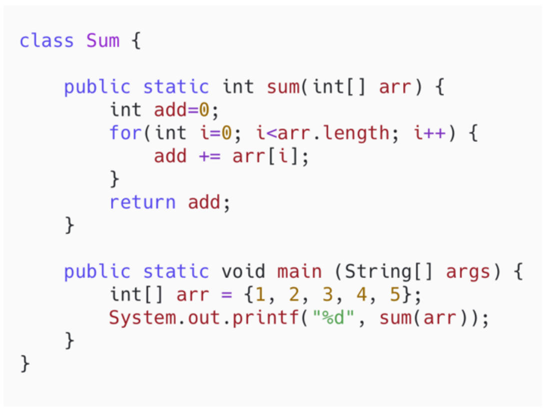
After many years of research on program comprehension, a variety of ideas, models, and tools have arisen that characterize how programmers think when they are attempting to comprehend source code. These have been developed to aid in the process of writing programs. In those methods Eye-Tracking has been a promising method to understand the program comprehension. An Eye-Tracking study by Sharif and Maletic (2010) which focuses on effect of identifiers naming conventions (camelCase and under_score) on program comprehension in which they found that significant improvement in time and lower visual effort with the underscore style [6]. Another Eye tracking study by Peitek et al. (2018) in which Eye tracking is added with fMRI to obtain more comprehensive understanding of program comprehension [3]. The Eye-Tracking study for programming style iteration and recursion by Aqeel at al. (2021) also been done which helps us to understand which programming style effect more in program comprehension [8]. Moreover, eye tracking study by Peitak at al. (2021) in which effect of linearity of source code and program comprehension strategy (top-down and bottom up) on linearity of reading order is studied [9]. After reviewing all the previous Eye tracking studies, we can say that there is no study has done about the effect of curly brackets in program comprehension. As curly brackets have been a crucial part of most of the programming language, so it is necessary to conduct a study regarding curly brackets.

There is different type of variation where we can use different style of curly brackets in the term of placing and use of them. During this research we will go through mainly three styles of curly brackets in program. One type will be where we skip the curly bracket especially for one-line statements. Second, we are going to start the curly brackets on same line. Third, we are going to use the curly brackets on next line. We call them shortly with -out -curly (WOC), with-curly (WIC), curly-new-line (CNL). We can see all three categories of curly brackets in Figure 1 in which they are demonstrated with the help of code snippet Array sum. Figure 1(a) an example of without curly brackets. Figure 1(b) an example of with curly brackets and Figure 1(c) is the example for curly brackets new line.



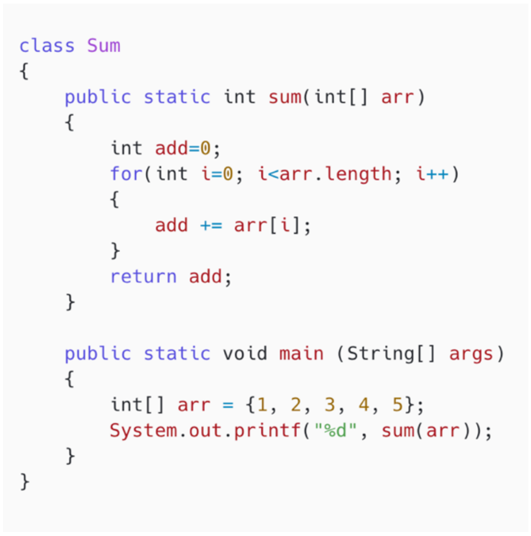
```java
class Sum {

    public static int sum(int[] arr) {
        int add=0;
        for(int i=0; i<arr.length; i++)
            add += arr[i];
        return add;
    }

    public static void main (String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

(a)



```java
class Sum {

    public static int sum(int[] arr) {
        int add=0;
        for(int i=0; i<arr.length; i++) {
            add += arr[i];
        }
        return add;
    }

    public static void main (String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

(b)



```java
class Sum
{
    public static int sum(int[] arr)
    {
        int add=0;
        for(int i=0; i<arr.length; i++)
        {
            add += arr[i];
        }
        return add;
    }

    public static void main (String[] args)
    {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

(c)

Figure 1. Java snippet to compute Array sum with different style of curly brackets

To comprehend the program accurately and precisely depend upon so many factors source code, programmers, identifier names [9][10]. Moreover, syntax of the programming language has been most essential part for understanding a program correctly. The motivation behind this study is to understand the behavior of the programmers for Programming styles (Different style of curly brackets). Curly brackets has been a huge part of many programming languages and plays significant role in comprehension of the program. There has been a huge debate regarding the use of curly brackets, especially in the context of position of the curly brackets [12]. If the use of curly brackets in different ways can be significant to the speed of code comprehension this could be vital impact on overall programming comprehension.

To achieve the appropriate and precise results for the study we are going to use an Eye-Tracker with PsychoPy which is one of the most vital approaches to measure program comprehension deeply.

1.2 Research Objective, Questions

The main objective of this research to investigate the programmer's comprehension when they are going to go through the different style of curly brackets. We are going to evaluate the effect in behavioral data and visual data and how it makes difference in these different conditions.

Based on above objective the following research questions are addressed in this thesis-

RQ1- Does the use of curly brackets in different styles makes any difference in the term of response time and correctness?

RQ2- Does the use of curly brackets in different style makes any difference in the term of Visual attention?

1.3 Research Significance

Sometime small problem can lead to big consequences in source code. How the only two-line code where pre-increment and curly bracket were omitted carouse a costumer sensitive information leak [10][11]. Therefore, significance of curly brackets cannot be ignored. As, Evolution of programming languages proceed the use of curly brackets also proliferated. The strong contribution of Curly brackets in programming languages like C, C++, Java, Rust, Swift etc. has increased. In C style like programming language curly brackets are used in many purposes like to define the scope of variable. Scope rules define the use of variables and extent of its visibly. It does not play any role beyond its scope [13]. In C, the usage of blocks is used to organize and compartmentalize code. A code block is a logically linked series of program statements handled as a unit. In C, a code block is formed by enclosing a sequence statement between opening and closing curly brackets. Many algorithms may be implemented with clarity, beauty, and effectiveness using code blocks. Furthermore, they aid the programmer in better understanding the real nature of algorithms [13][14]. In addition, After C, C++ which solved the complexity problem, Java language inherited the legacy of C language. Java comes with a unique feature of portability and OOP (Object Oriented Programming). Where the entire class definition including all its member belongs between opening curly braces ({) and closing curly braces (}). Curly brackets used also to define the block of code, method, and local space. The features like Encapsulation, Polymorphism, Inheritance also use curly brackets [15]. As, the curly brackets have been a vital part of programming languages, it is necessary to investigate the effect of these while using different style of programming based on position of curly brackets.

1.4 Limitations

The study has potential limitations. As our study focuses on use of Curly brackets in different style. As the study is concern so far is valid for the programming languages which has the use of curly brackets in syntax basis, e.g., C, C++, Java, JavaScript, Rust, Groovy, Kotlin, Perl, PHP, Scala, Swift. On the other hand, programming languages like Python which uses indentation to separate the different block of code. Although, Python also uses Curly brackets on to generate dictionaries, but our focus is on the Positional uses of curly brackets. Therefore, our results will not be applicable for programming languages like Python.

1.5 Research structure outline

The contents of each chapter in this thesis report are highlighted in this section. The following is a summary of each chapter:

- **Chapter 1 (Introduction):** The thesis work is introduced in this chapter. This chapter provides information on the objective, questions significance, limitations, and thesis structure.

- **Chapter 2 (Literature review):** Current chapter will give an overview of earlier studies that are pertinent to this thesis topic. It provides information on programme comprehension specifics, basic structure of programming, use of curly brackets in programming, methods for testing programme comprehension.

- **Chapter 3 (Methodology):** This section provides the information about research design, research philosophy, data collection method, data analysis method, limitations of the design.

- **Chapter 4 (Results):** This section provides the information on how the results are interpreted by data analysis method.

- **Chapter 5 (Discussion)**: An overview of the findings and a discussion of significant study components are presented in this chapter.

- **Chapter 6 (Conclusion)**: This chapter offers the study's conclusion and discusses the topic's potential future research application.

2 Literature Review

2.1 Brief history of Curly brackets

All current programming languages use programming blocks in their syntax. However, blocks were not included in the first high-level programming language. Fortran [45], which debuted in 1950, does not support compound statement (blocks). The function is declared on a single line, rather than in a block of code. Later that year, in 1950, a new language named ALGOL [46][47] was established, which featured blocks for the first time. The ability to arrange statements into compound statements known as blocks is one of Algol's characteristics. Algol used the keywords "begin" and "end" to indicate the beginning and end of the block. A block may be nested within another block, with the outer block acting as the dominant block and the inner block acting as a subordinate block.

BCPL [48], created by Martin Richards at the University of Cambridge in 1967, was the next shift in the syntax of block-structured language. The reference to BCPL is offered since C was created as a refinement and upgrade of BCPL by Ken Thompson's intermediate language named B. Because BCPL was designed to be a system development language, it was efficient as an assembly language; nevertheless, syntax must be at a relatively high level to make coding simpler and more productive. This implies that many elements of the high-level language ALGOL must be included into languages such as BCPL, although in a more efficient way.

To do this, Richards used the symbol $(for opening and) $ for closing, rather than (begin and end) for each block of code. Around 1969, Ken Thomson and Dennis Ritchie at Bell Labs started experimenting with alternative techniques to create an operating system using a system programming language. Thompson first attempted to do it in Fortran, but soon abandoned the concept due to technical difficulties. He decided to change BCPL to make system programming simpler, and so B [49] was born. Even though B was closer to the system development language, it still did not suit their needs. As a result, they created another language known as NB (New B) but it did not last for long time and was soon replaced with an entirely new language, which they created which is called “C".

Thomson rectified several items in B that carried over into NB and subsequently C[14], including shorter operators. These were required for the extended language to fit inside the memory restrictions of the computers of the day. Thompson, for example, devised a compound assignment operation (+=), as well as the increment (++) and decrement (--) operators, to improve the language's efficiency. Other BCPL operators were simplified because of this efficiency shift, such as $ and $ being replaced with “{“and “}”.

The curly brackets are a required symbol for blocks in several programming languages, including C++, Java, C#, and JavaScript, which closely follow the C-style. Curly braces have been adopted by more fascinating modern languages. To include GO and Rust [16].

2.2 Syntax and semantics in programming language

Language enables people to communicate via the use of sounds and symbols that are written down. Language is something that people pick up as a byproduct of their experiences in life, but in linguistics, which is the scientific study of languages, the structures and connotations of words are put through a more in-depth analysis. This field of study is also applicable to the topic of this text, which is computer programming languages. Programming languages, in contrast to natural languages, with which we communicate our thoughts and feelings, can be seen as artificial languages defined 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.
Many of the techniques and terms used in linguistics are also used in programming languages. For instance, language definitions are made up of mainly two parts syntax and semantics [51].

2.2.1 Syntax

A programming language's syntax is used to represent the structure of programmes, but the meaning of the programmes themselves is not taken into consideration. It entails the usage of a set of rules that verifies the sequence of symbols and instructions that are used in a programme. In general, rewriting rules are what make up Grammars, and their job is to produce programmes. Grammar is comprised of a limited number of grammatical categories, individual words, and rules that are already formed. These formal and informal methods may be used to understand the syntax of a programming language [50] :

- Lexical syntax

  The rules for fundamental symbols like as identifiers, literals, punctuators, and operators are defined with the help of these syntax.

- Concrete syntax

  The lexical units, also known as tokens, of a computer language are described by this syntax. It more focuses how the expression looks.

2 + 3	-- infix
(+ 2 3)	-- prefix
(2 3 +)	-- postfix
the sum of 2 and 3	-- English

- Abstract syntax

  This syntax merely delivers the most important information about the application. We can see in figure that sum expression has two operand expression in its significant part.

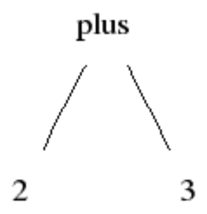
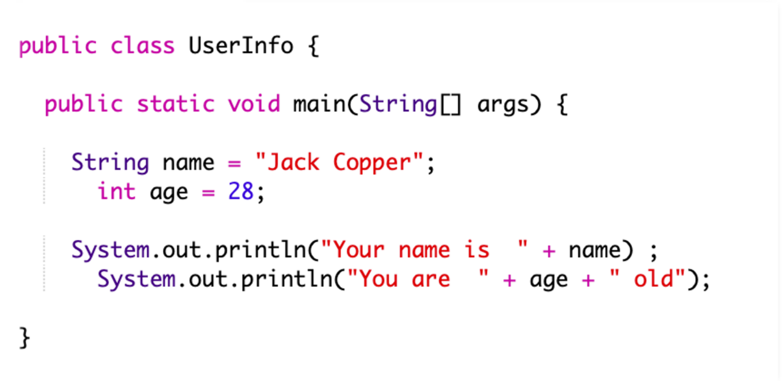


Figure 2. Abstract syntax example

2.2.2 Syntax Error

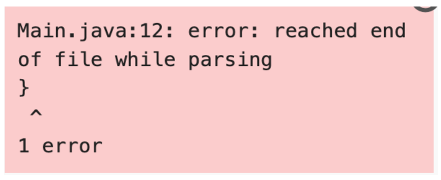
A mistake in the syntax of a coding or programming language that was input by a programmer is referred to as a syntax error in the field of computer science. A piece of software known as a compiler is responsible for finding syntax errors in a programme. The errors need to be fixed by the programmer before the programme can be built and then executed. Because programming languages are designed to be highly precise and free of ambiguity, we make a syntax error if we fail to respect or adhere to the vocabulary of the language. Because of this, the software will not be able to execute, and instead, a helpful error message will be printed.



```
public class UserInfo {

  public static void main(String[] args) {

    String name = "Jack Copper";
      int age = 28;

    System.out.println("Your name is  " + name) ;
      System.out.println("You are  " + age + " old");

}
```

Figure 3. Syntax Error example

The function above will not compile, and it will give us an error massage as shown in figure a. Syntax errors are handled at the compile time so it will give us compile error. The reason of this error in above code is just a missing curly bracket. It would have given us error any way if we miss any syntax of the language as in Java semicolon, curly brackets, variable definition are one of the most syntax to work with.



```
Main.java:12: error: reached end
of file while parsing
}
 ^
1 error
```

Figure 4. Syntax Error message

2.2.3 Semantics

The semantics of a language explains the meaning of syntactically correct sentences. In the case of natural languages, this entails associating certain words and phrases with particular things, ideas, and emotions. Semantics is the study of how computers act when given instructions in a programming language. This behavior might be shown, for example, by providing a detailed description of how a programme would run on a concrete or hypothetical computer, or by demonstrating the connection between the input and output of the code. When trying to understand how the syntax and the model of computation relate to one another in a programming language, the word "semantics" is invaluable. It places an emphasis on the interpretation of a programme such that its results may be easily predicted or understood by the programmer. Syntax-directed semantics is utilized to use a functional mapping of syntactic constructions to the computational model. Algebraic semantics, axiomatic semantics, operational semantics, and denotational semantics are some of the methods that may be used to explain the semantics of a programming language [52][53].

- Algebraic semantics

In algebraic semantics, information and linguistic structures are specified using algebraic notation. The goal of the algebraic approach to semantics is to provide a formal, axiomatic description of the attributes of various types of objects and operations on those objects.

- Operational semantic

The idea behind operational semantics is to be able to express the meaning of a programme beginning from a particular state by looking at its end result, which is the state in which the memory is left after the execution of the programme. This can be done by looking at the memory's state after the programme has been run.

- Denotational Semantics

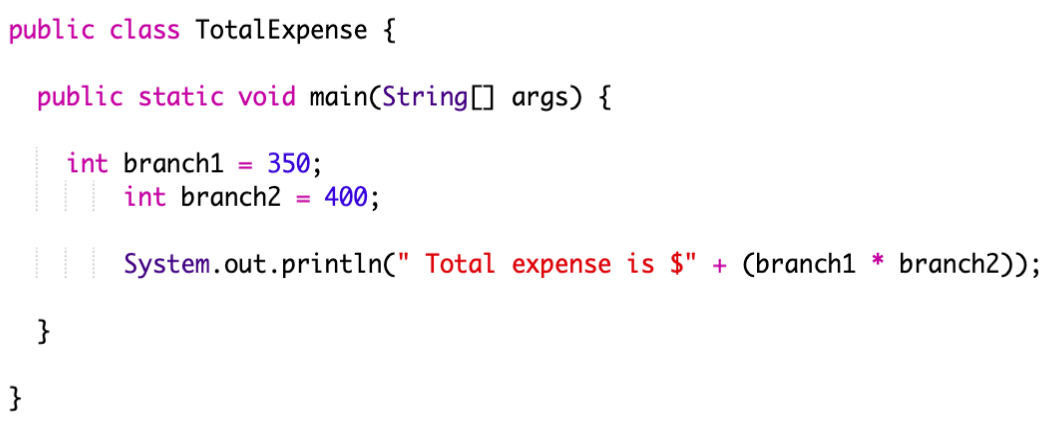
This semantics is based on the assumption that a programme may be seen in the same way as a mathematical function, that is, the impact of a programme can be viewed as a mathematical function in state.

- Axiomatic Semantics

The topic of Axiomatic Semantics is whether or not a particular programme is partially right (with regard to pre- and post-condition). In order to ascertain the intended purpose of an application, it constructs assertions about an association to be checked at each stage of the program's execution (i.e., implicitly).

2.2.4 Semantic Error

Even if your programme is successfully compiling, it may still be difficult to get it to generate the results you want from it. When a statement is correct in terms of syntax but does not have the effect that the programmer intended, this is an example of a semantic error. A mistake in meaning is referred to as a semantic error, as contrast to a syntax error. If a programme has this sort of problem, it will be able to execute, but the result it produces will not be accurate. We will understand it with an example of snippet where we mistakenly input a wrong athematic operation.



```
public class TotalExpense {

  public static void main(String[] args) {

    int branch1 = 350;
        int branch2 = 400;

        System.out.println(" Total expense is $" + (branch1 * branch2));

  }

}
```

Figure 5. Semantic Error example

This program will output Total expense is $140000. This is incorrect; the expected output is $750. This happens because we use a multiply (*) sign instead of an addition (+) sign. We can see the quick view of syntax and semantics with a table:

Comparison Basis	Syntax	Semantics
Basic	Permitted phrases of language	Interpretation of phrases
Error	Handel at compile time	Confronted at run time
Relation	Syntactic interpretation must have some distinctive meaning	Symantec component is associated with synthetic representation.

Table 1. Syntax and semantic comparison.

2.3 Curly brackets in Programming language

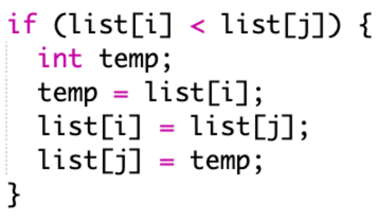
We have discussed in section 2.2 how curly brackets has come into play into programming language and how programming languages evolved with their syntax from Fortran to Rust. In this section we are going to discuss the different usage of curly brackets in programming language and how it has played vital role in programming language for long time [54].

2.3.1 Scope

Scope is an essential consideration while learning about variables. A variable's scope is the set of statements in which it may be used. If a variable may be referenced or assigned in a given statement, it is said to be "visible" in that statement. How a given name is linked to a variable, or in the case of a functional language, an expression, is determined by the language's scope rules. In particular, scope rules specify how the declarations of variables outside the present subprogram or block are linked to references to those variables (blocks are discussed in Section 2.2.2). When a variable is only used inside its specified scope, it is said to be "local" to that section of code.

2.3.2 Blocks

The C-based languages enable declarations to be included in any compound statement (a series of statements enclosed in matching braces) to establish a new scope. We refer to these combined statements as blocks. If list were an integer array, one may write as an illustration:



```
if (list[i] < list[j]) {
  int temp;
  temp = list[i];
  list[i] = list[j];
  list[j] = temp;
}
```

Figure 6. Example of block of code

In figure e: code between opening and closing curly brackets is a block of code and the scope of the *temp* variable is defined inside the block. As we can see how curly bracket is vital for one of the most important concepts of programming.

2.3.3 Use of curly brackets in Programming language

The C and C++ and Java programming languages rely heavily on curly braces (sometimes known as "braces" or "curly brackets"). There are different methods to denote the beginning and end of a programming structure, such as a loop, procedure, or conditional expression, in various programming languages. For instance, Java and C++ are sometimes referred to as curly brace languages since curly braces are used to designate the beginning and end of a code block. We will also only focus on only some significant parts of programming language where curly brackets used which are common for most of the languages. There has been huge significance of curly brackets in programming languages we are going to discuss some of them [55][56]:

a) Functions

A function is essentially a piece of code that you can use again and over again, rather than writing it out several times. Functions allow programmers to break down or divide an issue into smaller segments, each of which performs a specific purpose.

```
public class Main {
  static void myMethod() {
    System.out.println("Statement");
  }

  public static void main(String[] args) {
    myMethod();
  }
}
```

Figure 7. Function example

Here is the example of simple Java function which has a function name myMethod which is inside a class name Main. We can observe here that after defining a class name and function name we have to use curly brackets to construct a function and in the end of function there should be closing curly brackets.

b) Loops

A loop is a set of instructions in computer programming that is executed over and over again until some predetermined condition is met. While loop will repeat the statements within their related block of code until the expression inside the parenthesis turns false.

```
var = 0;
while(var < 200) {
  // do something 200 times
  var++;
}
```

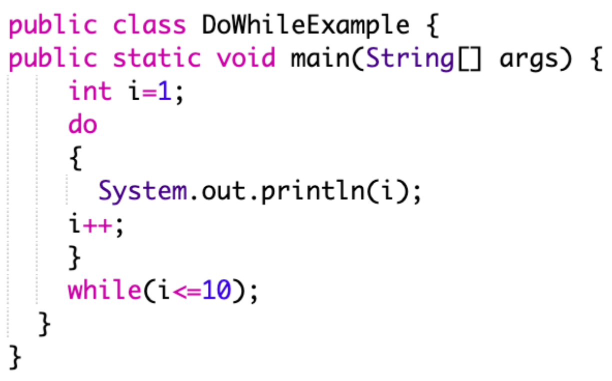
Figure 8. Example of while loops

If you need to construct a loop that runs a certain number of times, you may do it quickly and easily with the help of a control structure called a for loop. Whenever you have a certain number of times that you need to do an action, a for loop is a great method to have at your side.

```
public class ForExample {
public static void main(String[] args) {

    for(int i=1;i<=10;i++) {
        System.out.println(i);
    }
  }
}
```

Figure 9. Example of for loop

The Java do-while loop iterates across a block of code repeatedly until a certain condition is met. Use a do-while loop if you need to run the loop at least once and the number of iterations is not known in advance.

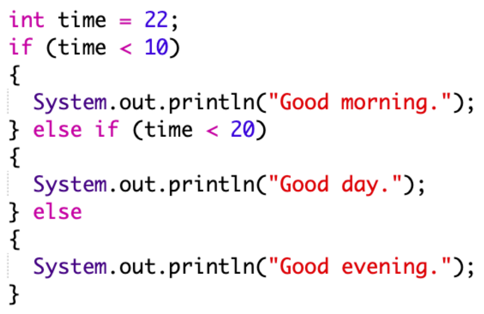


```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;
    do
    {
      System.out.println(i);
    i++;
    }
    while(i<=10);
  }
}
```

Figure 10. Example of do while loop

c) If/else statements

If a particular condition is true, the if/else statement runs a block of code. If the condition is not met, another block of code may be run.

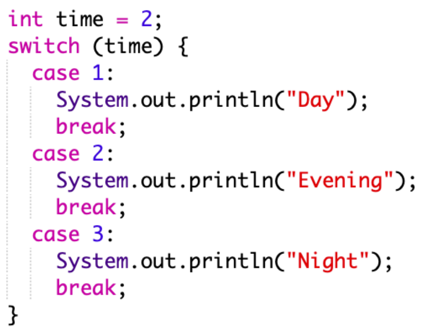


```
int time = 22;
if (time < 10)
{
  System.out.println("Good morning.");
} else if (time < 20)
{
  System.out.println("Good day.");
} else
{
  System.out.println("Good evening.");
}
```

Figure 11. If/else statements example

d) Switch statements

The switch statement is a kind of branch statement called a multi-way branch statement. Putting it more simply, the switch statement in Java allows you to execute one statement based on many sorts of conditions.



```
int time = 2;
switch (time) {
  case 1:
    System.out.println("Day");
    break;
  case 2:
    System.out.println("Evening");
    break;
  case 3:
    System.out.println("Night");
    break;
}
```

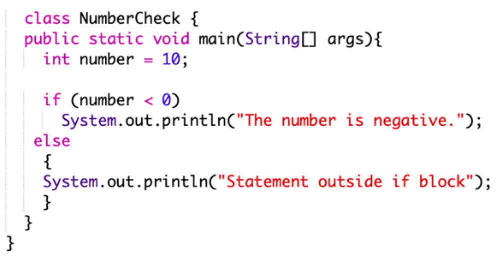
Figure 12. Switch statements example

We have seen so many examples of the use of curly brackets in programming language. Curly brackets are almost everywhere which makes them significant.

There are other uses of curly brackets as well for example defining an array.

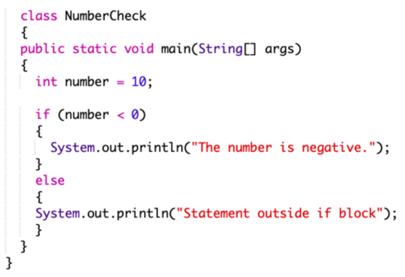
2.3.4 Placement of curly brackets

We have seen in section 2.2.3 that how curly brackets are used widely in programming languages but one really important question about those is which placement of the curly brackets is important for programmer to understand the program better. Using proper programming style may lead to the creation of superior software. A well-written style makes source code easier to read and comprehend, which in turn may minimize the number of mistakes and make it simpler to maintain [57]. There might be three possible placement conditions of curly brackets. First, placing curly brackets from same line. Second placing curly brackets from next line and last omitting curly brackets when not needed especially for single statements. There have been many thoughts on the placing of curly brackets. Some believe that they want to use curly brackets in every condition even though that is single line statement or not, they think with this readability and maintainability of code maintained [58]. Some people, believes that starting curly brackets from new line makes code cleaner and easier to understand [59]. One more suggestion which focuses on using brackets from same line as it reduces the line of code [60].



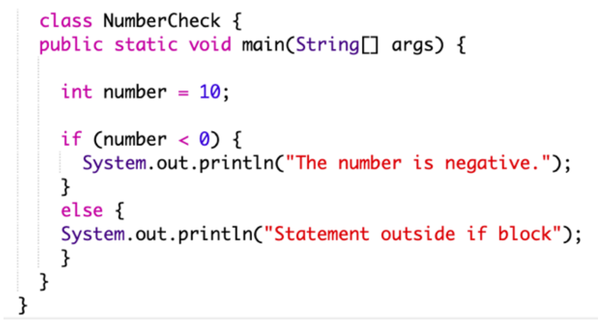
```
class NumberCheck {
public static void main(String[] args){
  int number = 10;

  if (number < 0)
    System.out.println("The number is negative.");
 else
  {
  System.out.println("Statement outside if block");
  }
 }
}
```

(a)



```
class NumberCheck
{
public static void main(String[] args)
{
  int number = 10;

  if (number < 0)
  {
    System.out.println("The number is negative.");
  }
  else
  {
  System.out.println("Statement outside if block");
  }
 }
}
```

(b)



```
class NumberCheck {
public static void main(String[] args) {

  int number = 10;

  if (number < 0) {
    System.out.println("The number is negative.");
  }
  else {
  System.out.println("Statement outside if block");
  }
 }
}
```

(c)

Figure 13. Example of curly brackets

We can see Figure 13 (a) shows the snippet missing curly brackets, Figure 13 (b) shows the code has brackets from new line at last, Figure 13 (c) using brackets in same line.

2.4 Program Comprehension

In the software development process, understanding the code is crucial. An improved knowledge of programme comprehension would aid in the creation of technologies and tools that streamline this process, saving both time and money. Comprehending a programme is a difficult cognitive process that can only be quantified via carefully designed studies. There are numerous approaches to measure program comprehension, here we discuss the following [17].

2.4.1 Think – Aloud Protocol

For almost a century, self-reporting techniques have been used to observe cognitive processes. One such way is to think aloud. Subjects are asked to express their sensations, feelings, and thoughts while doing the study. The session is either audio or video recorded [18].

Two roles exist in the Think-aloud method: observer and subject. The observer takes notice of the subject's spoken communication. In software engineering study, data is gathered from just one participant at a time; the number of individuals is often relatively small [19].

2.4.2 Memorization

Memorization tests provide participants a code fragment to explore for a certain time. Subjects are then asked to remember as much of code as they can. Allowing developers to memories source code to test their understanding may seem unusual today, but it's a vital component of the development process.

The researchers investigated the effect of priming on response time. Priming is the act of responding fast to a target stimulus if you have previously experienced a comparable stimulus. Participants were asked to recollect code snippets supplied by Soloway and Ehrlich to developers. The study discovered that skilled programmers relied more on coding standards such as sensible variable names[20][21].

2.4.3 Comprehension Tasks

Another job for programmers in the Soloway and Ehrlich research was to fill in a left-out line to properly finish the source code, referred known as "fill in the blanks." Developers can only complete the software properly if they first understand it. He measured correctness and response time of participants. Since this task explicitly requires participants to understand source code, it is more direct than requiring participants to memorize code. He found that different operators lead to different response times, and that statements that are true are processed faster [20][21].

2.5 New techniques to measure program comprehension:

2.5.1 Eye tracking:

In 1990s when Eye tracking was first used in software engineering. Researcher study model comprehension, program comprehension, debugging traceability. Eye-tracking technology in program comprehension is used by software engineering researchers to analyze the cognitive processes and efforts involved in various sorts of Software engineering activities. Using eye-movement data, an eye tracker (hardware and software) measures a participant's visual attention. Eye movements are vital to cognitive processes because they direct the participant's visual attention to the elements of a visual stimuli that the brain analyzes. Visual attention activates cognitive processes necessary for task completion. It is also a surrogate for visual effort, which is a subset of cognitive effort, and is assessed as the amount of visual attention assigned to various portions of a visual input. In software engineering experiments, the stimulus is shown on a computer screen [22].

There are several eye-tracking methods available for monitoring eye movement. Video-based tracking, infrared pupil-corneal reflection, and Electrooculography Scleral contact lens/ Search coil based tracking are the three most used ways. Pupil Center Corneal Reflection is the most prevalent method employed by contemporary eye trackers (PCCR). It employs a near-infrared camera or other optical sensor for gaze tracking. In this technique, near-infrared light is focused into the pupil, or the center of the eye, causing visible reflections in the cornea that are recorded by a camera. PCCR is most often used in remote, non-intrusive eye trackers. Eye trackers catch eye movement by illuminating the eye with a light source that creates visible reflections. It captures eye images with a high-resolution camera to display these reflections. Then, this eye picture is utilized to determine the light source's reflections on the cornea and pupil. The vector generated by the angle between the cornea and pupil reflections is then computed, and based on this data, the direction of gaze is determined [23].

2.5.1.1 Different kind of Eye tracking devices current time:

a) Screen based remote eye tracker:

These types of Eye Trackers require the respondents/participant to sit in front of a screen to interact with the stimuli or screen based content. Remote ET systems track the eyes within certain limits called headbox, but the eye movement freedom is sufficiently large and the respondents feel unrestricted. These eye trackers records eye movements at a distance. Unlike other Eye trackers participants do not need wear any equipment’s on head. These eye trackers are mounted to the computer screen. Participants sit in a front of computer screen to gather the eye tracking data.

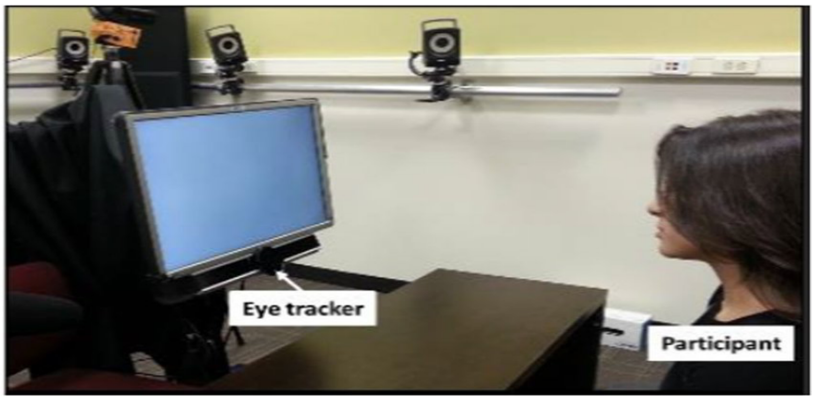


Figure 14. table mounted eye tracker[25]

b) Head mounted or mobile eye tracker:

These are placed in close proximity to the respondent's eyes and do not restrict their mobility in any way. These are used in the event that your research requires participants to carry out activities in a natural setting. On the other hand, glasses may move about while the recording is being done. These eye trackers are able to capture eye movements from a very near distance. These are often attached to frames that are made of a lightweight material. The responder is free to wander about while wearing these eye trackers, which is still another benefit of using them [23].

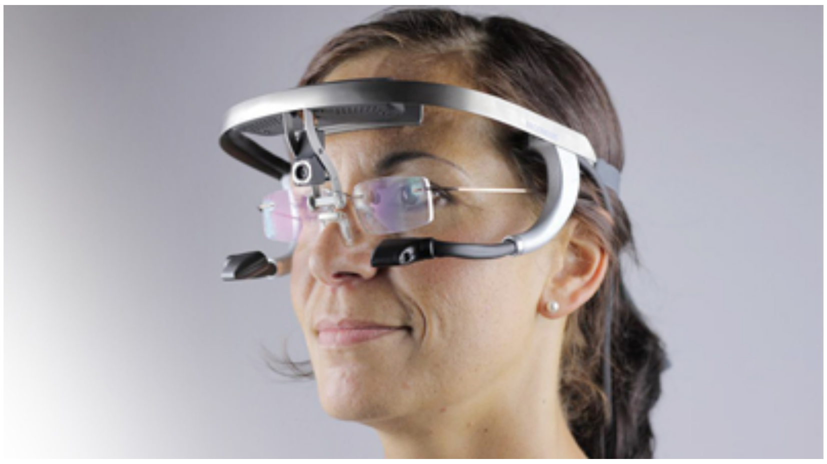


Figure 15. Head mounted Eye tracker

2.5.1.2 Terminology

Eye tracking systems capture gaze points that indicate where a participant is gazing on a stimulus. These gaze points may be used to infer information about the participant. Devices that are considered to be state-of-the-art are able to achieve speeds of between 60 and 500 Hz or even higher. The recording rate determines the number of gaze points that are captured in a single second. The following will provide a distinction between the various data kinds as illustrated in Figure 1, as well as information on each type.

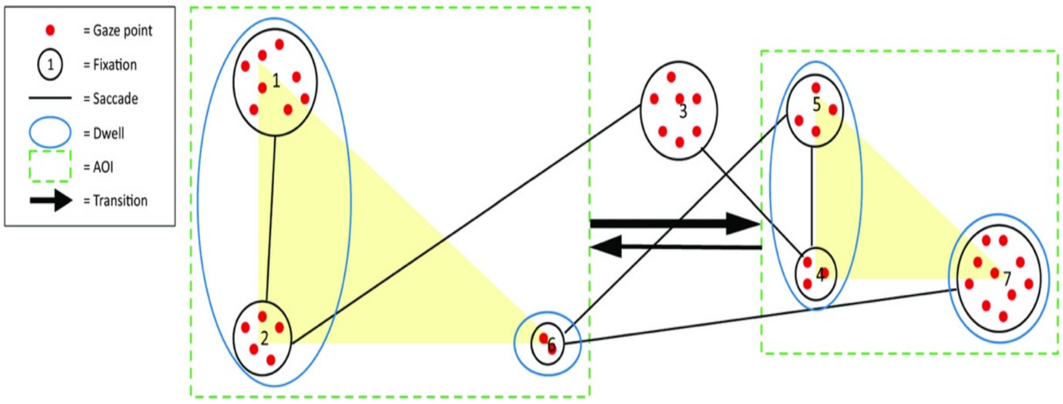


Figure 16. Demonstrates the Gaze points, Saccades, fixations. AOI(Area of interest), transition between AOI, and Scanpath.

- **Gaze points**

The basic units of measuring the eye movements are gaze points, one gaze point is one row captured by the Eye tracking devices.

- **Fixations**

Fixations are eye movements that stabilize the retina over an item of attention that remains motionless. A fixation is a cluster of gaze locations that are intended to be near in time and space. Fixation is the duration during which our eyes are fixated on a certain sensory item. Fixation time is typically between 100 and 300 milliseconds.

- **Saccades**

A quick eye movement that shifts the focus of the visual axis to a new position is referred to as a saccade. The phrase originates from an ancient French word that literally translates as "flick of a sail." Saccades are the quick eye movements that occur between fixations and have their own specific name. The length of a saccade may vary anywhere from 30 to 80 milliseconds, which is an amount of time that is sufficiently brief to make the executioner virtually blind throughout the transition. Most saccades are performed voluntarily. Micro-saccades, on the other hand, are short, jerky eye movements that are involuntary and occur during lengthy fixations to refresh the participant's visual memory.

- **Pupil Dealation**

The dilation of the pupil, which makes it possible for more light to enter the eye when the surrounding light levels are low. It may also occur if a participant's mood or attitude changes, as well as whenever they are asked to do complicated cognitive activities.

- **Scanpath**

The scanpath was first defined by David nation and Lawrence in 1971. A sequence of fixations that are recorded in the order in which they occurred and depict the participant's pattern of eye movement.

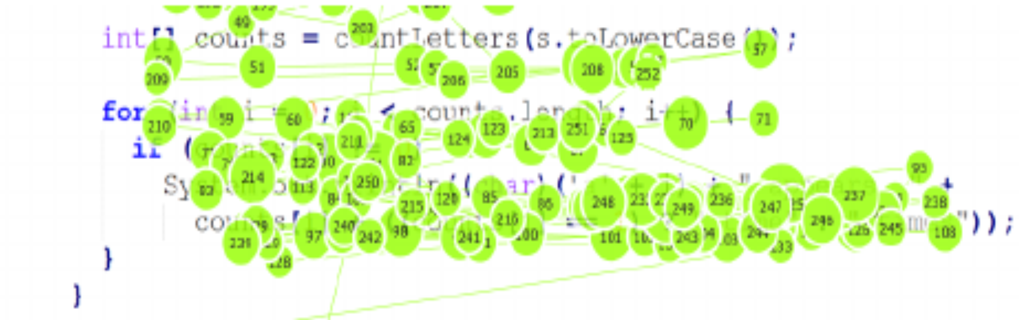


Figure 17. Scanpath on code snippet.

- **Heatmap**

A heat map is a visual representation of the distribution of visual attention that is generated by the accumulation of fixations and gaze locations, which may be either static or dynamic. Heat maps are a wonderful approach for visualization; they indicate the areas of the stimulus that have received the most attention. The color-coding technique used in heat maps is simple and easy to understand. The red region has a large number of gaze points, which indicates an enhanced degree of interest. On the other hand, the yellow and green areas signal toward less visual attention being paid to them.

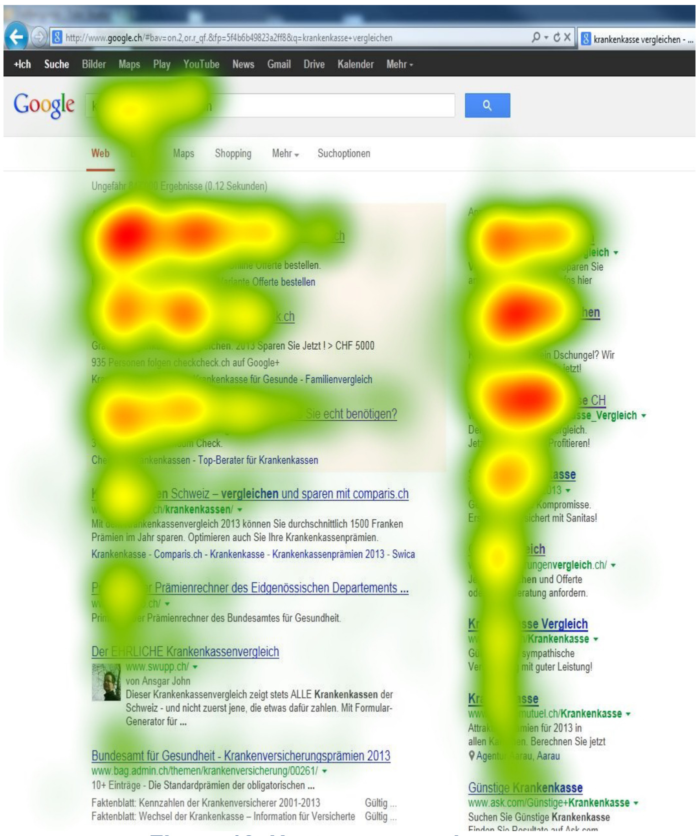


Figure 18. Heatmap example

- **Area of interest (AOI)**

  Areas of interest (AOI) are designated by the user as subregions of a stimulus item shown on the screen. Metrics for differentiating AOIs are based on the performance of two or more locations in the same image, website, or program interface [28][29][30].

2.5.1.3 Representative algorithms for saccade and fixations

This section describes the three two main algorithms to detect the fixations and saccades. There are many algorithms to detect fixation and saccades as we discussed in earlier section, but our goal will be detection fixation and saccades accurately without less noise.

- **Fixation detection (Dispersion – based algorithm)**

  The raw data exported from the eye tracker's software is used to figure out the fixation sequences. When this information isn't in the data, the dispersion-based threshold identification (I-DT) algorithm is used on the gaze data to find the fixations (Salvucci & Goldberg, 2000). The first step is to choose a window in which to figure out the dispersion D. We use a window size $\boldsymbol{W_{thresh}}$ of 50 ms as the minimum time for a segment to be considered a fixation. Using the minimum window size, W = $\boldsymbol{W_{thresh}}$, we can figure out the dispersion using Eq (1). Gx and Gy are the gaze position vectors on the x and y axes for a given window, and D is the window's dispersion. If D <= $\boldsymbol{D_{thresh}}$, the size of the window is increased by one, so W = W+1, and the last step is done again. Lastly, if D> $\boldsymbol{D_{thresh}}$ and W>$\boldsymbol{W_{thresh}}$, all the points in the window except for the last one is considered to be part of a fixation sequence. Then, the window is reset to Wthresh and starts at the first point after the last fixation sequence.

$$D = \sqrt{[\mathbf{max}(\boldsymbol{G_x}) - \mathbf{min}\ (\boldsymbol{G_x})]^2 + [\mathbf{max}(\boldsymbol{G_y}) - \mathbf{min}\ (\boldsymbol{G_y})]^2} \qquad (1)$$

After this fixation count, maximum duration, and average duration are extracted.

- **Saccades and microsaccdes detection (Velocity – based algorithm)**

When capturing data from an eye tracker, the program will often flag saccade sequences. The velocity-based threshold identification (I-VT) approach is used for data in which the saccade sequences are unlabeled. The procedure begins by determining the pointwise velocity of the gaze data. We call movements of points faster than around 40 pixels per second "saccades." Using gaze data, the velocities may be calculated as shown in Equation (2). V*t* represents the x or y velocity at time t, and G*t* represents the x or y gaze location at time t in the equation.

$$\boldsymbol{V_t = G_t - G_{t-1}} \qquad (2)$$

First, determine velocity from gaze data using a moving average (Equation) (3a). Calculate $\boldsymbol{V_{thresh}}$ using Equation (3b). All points with k > 1 are part of a microsaccade sequence if the sequence has at least six samples. The microsaccade sequence's velocity and amplitude are computed using Equation (3d & e). Last two equations compute saccade velocity and amplitude. (3): $\boldsymbol{V_t}$ is the x- or y-axis velocity at time t; $\boldsymbol{G_t}$ is the gaze location; $\boldsymbol{S_{freq}}$ is the eye tracker sampling frequency. $\boldsymbol{V_{thresh}}$ is the microsaccade threshold velocity in the x or y axis; $\boldsymbol{V_{fac}}$ is a constant value used to calculate the microsaccade threshold velocity (default value is 5); $\vec{\boldsymbol{V}}$ is the velocity vector in the x or y axis; $\vec{\boldsymbol{G}}$ is the gaze position vector in the x or y axis; $\boldsymbol{V_{peak}}$ is the microsaccade sequence peak velocity; and A is the microsaccade sequence amplitude.

$$\boldsymbol{V_t} = {(\boldsymbol{G_{t+2} + G_{t+1} - G_{t-1} - G_{t-2}})}\Big/{(\boldsymbol{6 * S_{freq}})} \qquad 3(a)$$

$$\boldsymbol{V_{thresh} = V_{fac}} * \sqrt{\boldsymbol{median}((\vec{\boldsymbol{V}} - \boldsymbol{median}(\vec{\boldsymbol{V}}))^2)} \qquad 3(b)$$

k = $(\overrightarrow{V_x}/V_{x_{thresh}})^2 + (\overrightarrow{V_y}/V_{y_{thresh}})^2$ 3(c)

$$V_{peak} = \max\left(\sqrt{\overrightarrow{V_x} + \overrightarrow{V_y}}\right) \quad 3(d)$$

$$A = \sqrt{[\max(\overrightarrow{G_x}) + \min(\overrightarrow{G_x})]^2 + [\max(\overrightarrow{G_y}) - \min(\overrightarrow{G_y})]^2} \quad 3(e)$$

In our study we have taken the help of these algorithms to detect the fixations and saccades.[32]

2.5.1.4 Different Eye Tracking Matrices

Researchers in SE, while planning an eye-tracking study, must choose appropriate metrics to quantify the visual effort that are indicative of the activities and stimuli being examined. The names, definitions, and applications of many visual-effort measures from prior research are provided. We classify metrics as either (1) fixation-based, (2) saccade-based, or (3) scanpath-based.

1. **Fixation based matrixes:**

- **Fixation Count**

  Each AOI's total number of fixations is referred to as its "Fixation Count" (FC). By dividing the total number of fixations by the total number of words in the text, the number of fixations may be made proportional to the length of the text.

- **Fixation rate**

  Fixation Rate (FR) is the number of fixations in the area of interest (AOI) divided by the area of glance (AOG). AOG can be the whole stimulus or another AOI to calculate the ratio of the total number of fixations in one AOI to all fixations or the ratio of fixations between two AOIs.

- **Average fixation duration**

  Average Fixation Duration (AFD) is the sum of the durations of all the fixations divided by the number of fixations.

2. **Matrix based on saccade**

- **Regression rate**

  Regressions Rate shows how often saccades of any length go backward. Few regressions are a sign of a good reader, so a high rate of regressions means that the participants have trouble reading and understanding a stimulus [29].

2.5.1.5 Eye tracking studies in Program comprehension

At the beginning of the 1990s, people in the software engineering field became interested in eye-tracking technology as a way to study the reading strategies that people use to understand programs. The first eye-tracking study in software engineering was done by Crosby et al. [Crosby and Stelovsky, 1990]. They investigated how different ways of reading affect the ability to understand procedural code.

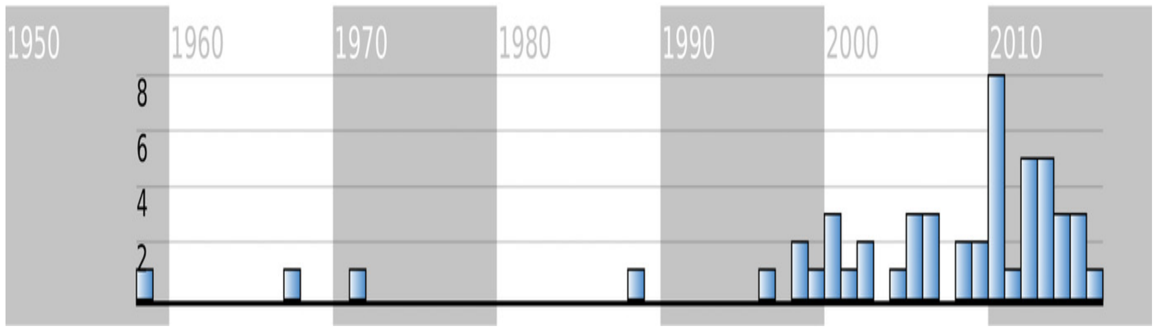


Figure 19. Histogram of publication of journal articles, conference paper and books in Eye-tracking

Figure 19 shows number of eye tracking studies published in recent years, and it has strongly increased in last decade. Some eye tracking studies in program comprehension are discussed below.

- [Bednarik and Tukiainen, 2006] gives a way to visualize Java source code and looks at how programmers use both the source code and the visualization [35].

Artifacts	Three Java source codes of factorial (15 LOC), recursive binary-search (34 LOC), and naive string matching (38 LOC)
Participants	14 students
Eye tracker	Tobii 1750
Variable (DV)	Time and attention switching
Variable (IV)	Code vs Visualization
Variable (MV)	Not mentioned

Table 2. Bednarik and Tukiainen study on code comprehension [33]

- [Busjahn et al., 2011] performs an experiment to investigate the differences between source code reading and natural text reading [36] .

Artifacts	Java source codes
Participants	14 (not mentioned the type)
Eye tracker	Tobii T120
Variable (DV)	Time, number of characters, and number of elements
Variable (IV)	Source code vs. natural language text, and different source code parts including: operator, key- words, identifiers, and numbers
Variable (MV)	Not mentioned

Table 3. Busjahn study for code comprehension [33]

- [Busjahn et al., 2014] studies attention distribution on code elements to differentiate experts' and novices' code reading strategies [33].

Artifacts	Eleven Java source codes
Participants	15 professionals
Eye tracker	Tobii T120
Variable (DV)	Time
Variable (IV)	Code elements (identifiers, operators, keywords, and literals)
Variable (MV)	Expertise

Table 4. Busjahn's study for code comprehension [33]

- [Rodeghero et al., 2014] conducts an eye-tracking study and use its findings to build a code summarization tool [40].

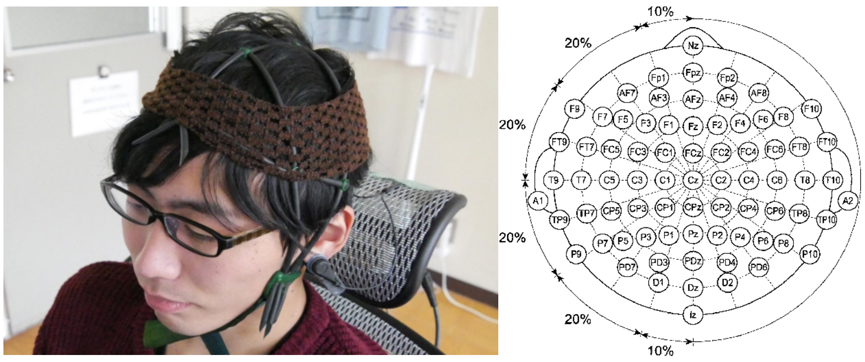
Artifacts	67 Java methods from six different applications: NanoXML, Siena, JTopas, Jajuk, JEdit, and JHotdraw
Participants	10 professionals
Eye tracker	Tobii TX300
Variable (DV)	Fixation number and duration, Fixation Time, and Number of regressions
Variable (IV)	Task difficulty (easy and hard)
Variable (MV)	Not Mentionen

Table 5. Rodeghero study for code comprehension [33]

We have discussed most of the studies done to understand program comprehension. We have mostly included the studies which uses Tobii as an Eye tracker because in our study we are also going to use Tobii eye tracker to understand program comprehension.

2.5.2 EEG(Electroencephalogram)

An EEG is a device which works on electrical activity that originates in the brain and is recorded using electrodes put on the scalp. The electric potential of the brain is measured as the difference between two electrodes. Each electrode is put on a position designated by the International 10-20 system, which is shown in Figure 1. Except for the electrodes defining ground potential, the 10- 20 system specifies 19 electrode locations [41].



(a) (b)

Figure 20. a) Participant wearing EEG (b) International 10-20 system of electrode

2.5.3 fMRI (Functional magnetic resonance images)

Since 1991, researchers in neuroscience have used functional magnetic resonance imaging (fMRI) to study cognitive processes. FMRI is based on monitoring changes in blood. The findings may be utilized to deduce which brain regions are involved cognitive functions.

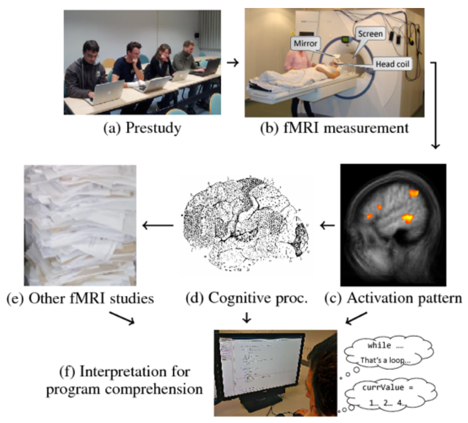


Figure 21. shows workflow of fMRI [42]

Norman Peitek and Janet Siegmund have provided the workflow of fMRI in their study. That we can see through the image 6. Where they have studied program comprehension with fMRI [42].

3 Methodology

As there has been lack of research in the term of curly brackets in program comprehension which we have discussed in section 1.1. So, it is important to study curly brackets in programming in a stronger approach. We have already discussed about our aim and objective that we are doing to see how the different style of curly brackets makes any difference on response time, correctness, and visual attention. In this chapter we are going to focus on our research plannings to achieve the required results we need.

3.1 Research planning

Planning experiments ahead of time is essential for achieving reliable results. Having a clear research objective and well-crafted experiment materials are crucial in the planning and conceptualization stages of any experiment. In this chapter, we will look at some of the additional factors that should be considered while designing an experiment. We have planned our experiments in some following steps. Research material, Participants for research, Tools and technology used, Research design, Analysis procedure.

3.1.1 Research Material

Code snippets are presented as experimental materials in this study. Code snippets are often the decisive element in the success or failure of an experiment designed to assess participants' program comprehension. Inappropriate code snippets for a certain research goal would provide inaccurate findings. Here, we provide a high-level overview of how the study's code snippets were created and selected.

3.1.1.1 Snippets Generation

Code snippets are generated in Java. We have chosen Java because it mostly covers all the vital properties where we can explore curly brackets. Code snippets are generated in the way that it can be reused for other experiments in future as well. Before we have generated 12 code snippets and because certain snippets did not meet our standards due to their length and complexity later, we have selected only 8. Total 8 code snippets are generated are following- ArraySum, BubbleSort, NumberCheck, ContainsSubstring, Power, BinaryConversion, MultiplyMatrix, Prime number. There has been limit of lines to create code snippets, maximum 35 lines. The code snippets are created in two parts and main three categories. Two vital parts are, First the calculation part of snippet and second the whole snippet with main function. Three categories of the snippets are with-out-curly (WOC), with-curly (WIC), curly-new -line (CNL).

So, we have generated total 48 snippets. We will know more about our snippet generation in section 3.3.4. We have used IntelliJ IDE to create code snippets. All snippets have been checked for errors and correctness.

Snippets	Without curly		With curly		Curly new line	
	LOC	Response Time(s)	LOC	Response Time(s)	LOC	Response Time(s)
ArraySum	12	41	13	64	17	64
BubbleSort	17	91	19	115	25	74
NumberCheck	19	65	21	51	27	74
ContainsSubstring	25	75	26	91	33	92
Power	13	66	14	68	16	48
BinaryConversion	17	76	18	52	23	86
MultiplyMatriix	24	83	26	84	34	113
PrimeNumber	22	51	24	51	32	51

Table 6. All snippets used in this study and their corresponding response

3.1.1.2 Snippets Selection

After the snippets generation we need to select the snippets for our study. Snippets selection process is necessary because snippets are our base of study. Inappropriate selection of snippets may lead to flaws in study. To select the snippets appropriately we have defined some criteria as followings-

- First, snippets should be easy to understand, our focus is on structure of the code instead of difficulty level. So, the snippets should be easy not extremely complex.

- The snippets should reflect well-known ideas covered in the early stages of the programming course. More sophisticated notions may perplex the participants.

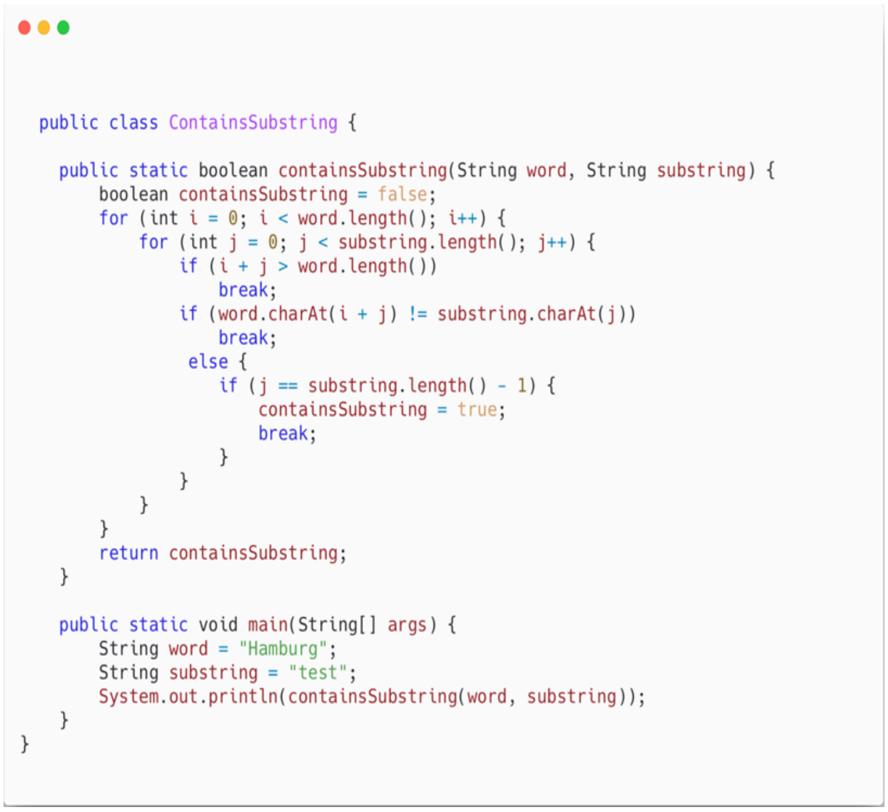
- As the snippets should be easy, at the same time snippets should be challenging enough. The scenario should not happen that participant has solved the problem just looking at it once. Snippets should be bit challenging that we can record the eye movements for our study.

- As our study is on effect of curly brackets so, snippet should contain operations like, for loop, if statements, OOP (Object-Oriented Programming) and functions. With this selection criteria we can explore curly brackets more strongly because all these operations need curly brackets to be used in programming language.

- Length of the code snippets must not be greater than 35 lines, this is because we do not want to give trouble of scrolling to the participant while doing the study

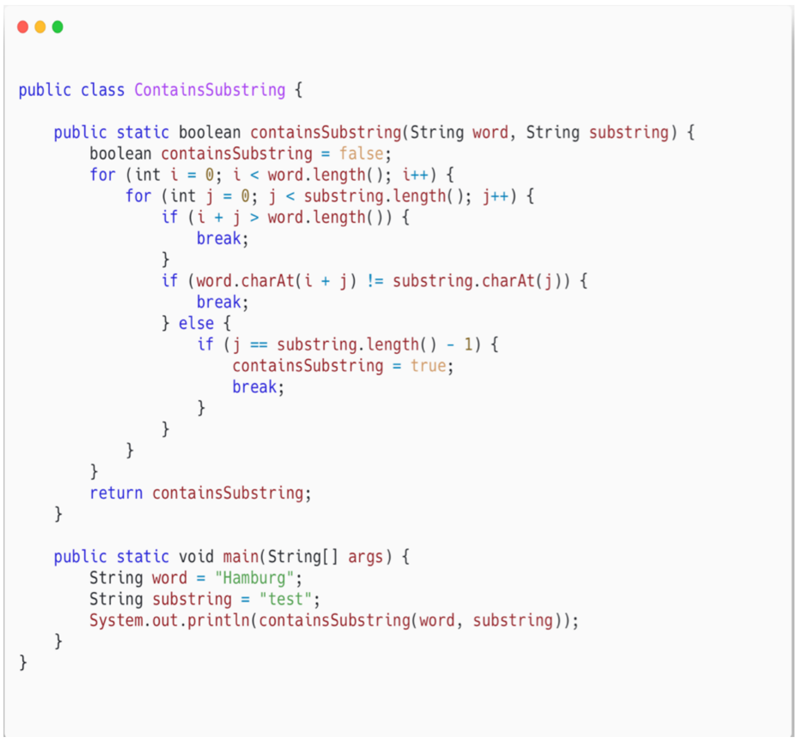
3.1.1.3 Code snippet Example

We are going to show an example of code snippets we have selected for our study. We can see the snippet example what we have used for our study. Figure 22 is the example of missing curly brackets when there is no need of them. We call them in research WOC (without-curly). Figure 23 shows our next type where we use all the brackets, but brackets start from the same line. We call them WIC (with-in-curly). And last the Figure 24 is the example of where we use curly brackets but from next line. We call them CNL (curly-new-line). We will discuss more about this in study design section 3.3.4.



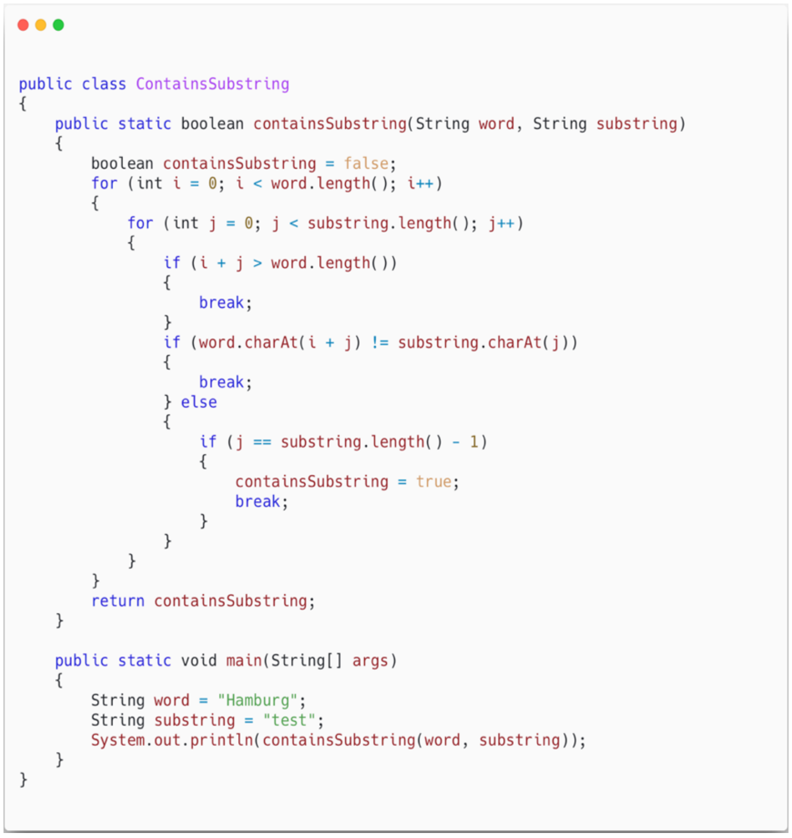
```
public class ContainsSubstring {

  public static boolean containsSubstring(String word, String substring) {
      boolean containsSubstring = false;
      for (int i = 0; i < word.length(); i++) {
          for (int j = 0; j < substring.length(); j++) {
              if (i + j > word.length())
                  break;
              if (word.charAt(i + j) != substring.charAt(j))
                  break;
               else {
                  if (j == substring.length() - 1) {
                      containsSubstring = true;
                      break;
                  }
              }
          }
      }
      return containsSubstring;
  }

  public static void main(String[] args) {
      String word = "Hamburg";
      String substring = "test";
      System.out.println(containsSubstring(word, substring));
  }
}
```

Figure 22. Snippet Example of ContainsSubstring (WOC)



```
public class ContainsSubstring {

    public static boolean containsSubstring(String word, String substring) {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++) {
            for (int j = 0; j < substring.length(); j++) {
                if (i + j > word.length()) {
                    break;
                }
                if (word.charAt(i + j) != substring.charAt(j)) {
                    break;
                } else {
                    if (j == substring.length() - 1) {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }

    public static void main(String[] args) {
        String word = "Hamburg";
        String substring = "test";
        System.out.println(containsSubstring(word, substring));
    }
}
```

Figure 23. Snippet Example of ConatinsSubstring (WIC)



```
public class ContainsSubstring
{
    public static boolean containsSubstring(String word, String substring)
    {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++)
        {
            for (int j = 0; j < substring.length(); j++)
            {
                if (i + j > word.length())
                {
                    break;
                }
                if (word.charAt(i + j) != substring.charAt(j))
                {
                    break;
                } else
                {
                    if (j == substring.length() - 1)
                    {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }

    public static void main(String[] args)
    {
        String word = "Hamburg";
        String substring = "test";
        System.out.println(containsSubstring(word, substring));
    }
}
```

Figure 24. Snippet example of ContainsSubstring (CNL)

3.1.2 Participants for research

The influence of curly brackets on programmer’s behavior is investigated in this study using data acquired from the study. The involvement of the appropriate participants are critical for the research to provide usable and meaningful findings. Inappropriate participants may provide undesirable findings and possibly lead the research to fail. The participants are chosen on the conditions that they have basic knowledge bout java programming. Knowledge of programming fundamentals, data structures, algorithms, and a basic comprehension of the java programming language is adequate to complete the survey's programming assignments.

Distribution	Participants data
Male	12/20(60%)
Female	8/20(40%)
Average age(years)	29
Average experience in java(years)	1.8
Average professional experience	2.1

Table 7. Demographic data of the participants

An online invitation sent for the participation in the study, if only they are interested in the study they come. Especially, university programming student groups was the source to find the participants. Before the study starts each participant has been informed about the study and how it works. Participants came to university and performed the test in university lab. We have included total 20 participants in our study.

3.1.3 Technologies and tools used

The experiment plane's execution necessitates the employment of a variety of tools and technology combinations. Each instrument or technology serves a specific purpose and operates at different phases of experiment implementation. Some of the most significant techniques and technologies employed in this research are as follows:

1. PsychoPy

We have used PsychoPy program in our study. PsychoPy is a program that enables users to design experiments in the field of behavioral science (which includes fields like psychology, neurology, and linguistics) with fine-grained control over the location and timing of stimuli. Users have the option of using either the traditional Python scripting interface or the brand-new graphical Builder interface to create their experiments. PsychoPy's main features are geared on managing the presentation and timing of stimuli.

- **Builder in PsychoPy**

The builder in PsychoPy allow user to create a graphical representation of an experiments. In PsychoPy Builder, an experiment is described by a set of routines. Each routine has one or more components, such as stimuli and possible responses. An image of the Builder interface can be seen in figure 25.

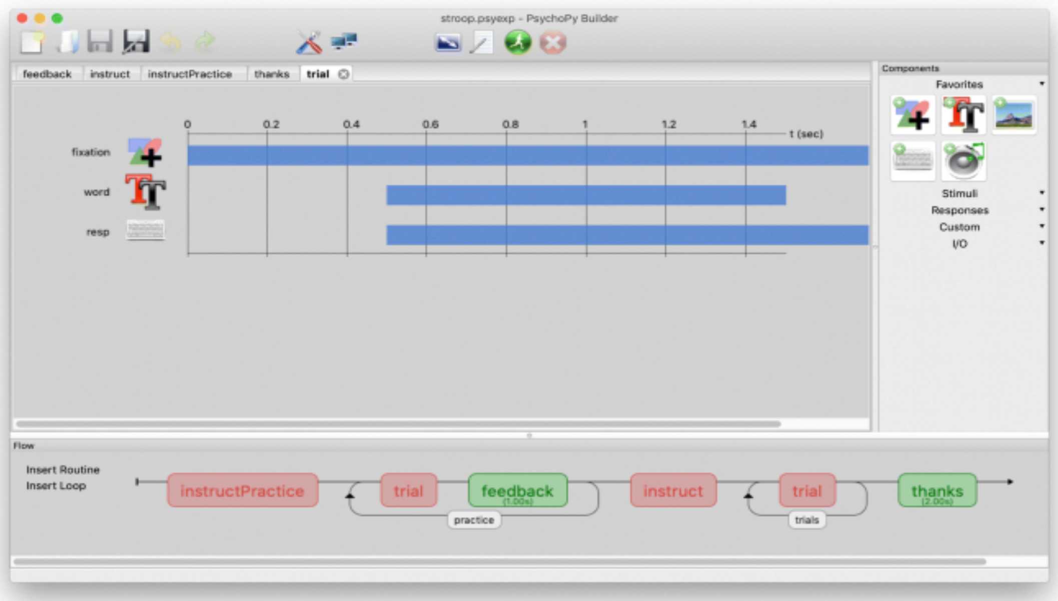


Figure 25. Builder interface PsychoPy

The way PsychoPy builder talks to you. The components that can be added to the experiment are listed in the right-hand panel. They are grouped into categories that can be expanded or shrunk. These components can be put into routines and show up in the routine panel as "tracks." In this demo, in the routine named "trial," we just show a word after milliseconds pause and start watching the keyboard for responses at the same time. However, any number of components can be set to start and stop at the same time or at different times. The bottom panel of the interface shows the "Flow" of the experiment. This is the order in which the "Routines" will be shown, as well as any "Loops" that can be used to repeat trials and/or blocks and control the randomization of conditions. Users say that this view makes putting their experimental designs into action very easy and gives them a lot of freedom.

The builder reaches its goal by being easy to use for teaching, flexible enough for high-quality experiments, and accurate and precise.

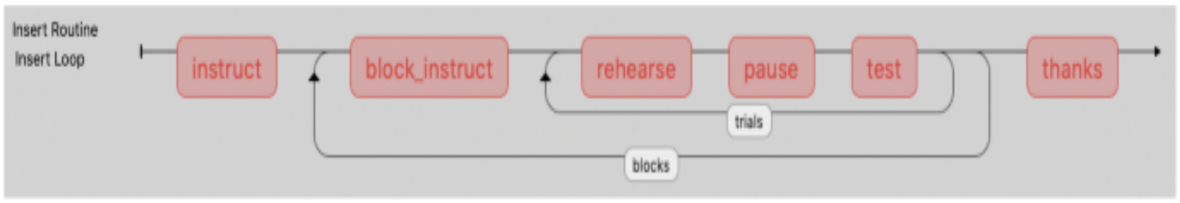


Figure 26. more complex flow arrangements

Figure 26 shows more complex flow arrangement. Loops and Routines can be nested in arbitrarily complex ways. PsychoPy itself is unconcerned about whether a Loop designates trials, a sequence of stimuli within a trial, or a sequence of blocks around a loop of trials, as above. Furthermore, the mechanism for each loop is independent; it might be sequential, random, or a something more complex, such as an interleaved staircase of trials [43].

2. Eye Tracker

We have used a table mounted Tobbi Pro Fusion Eye-Tracker in our study. We have discussed about Eye-Tracker in section 2.5. Where we have discussed Eye-Tracker functions, their types and function of different eye trackers.

3.1.4 Research Design

Keeping in mind that the purpose of this research is to investigate how different style of curly brackets influence programmers' behavior in terms of response time, correctness and visual attention during program comprehension, the following factors are described:

- **Dependent variables**

Dependent variables in our study are participant’s response time, correctness, and visual attention while program comprehension. Response time is a time that participant took to comprehend the snippet and time it took to click on the output. Correctness is a percentage of correct answer participant gave during the study. Visual attention is the factor that which tells us where and how long participant look to comprehend the snippet.

- **Independent variables**

Independent variables in our study are code snippets with different style of curly brackets (WOC, WIC, CNL). Without curly bracket (WOC) is a type of snippet where curly brackets are missing for where it is not so useful to use especially for single line statements. Within Curly bracket (WIC) is the type of snippet where all the curly brackets are used but all curly brackets start from same line. Curly bracket new line (CNL) is the type of snippet where all required curly brackets are used but each curly bracket start from new line.

The goal of this study can be expressed as a research question:

RQ1- Does the use of curly brackets in different styles makes any difference in the term of response time and correctness?

RQ2- Does the use of curly brackets in different style makes any difference in the term of Visual attention?

After defining the research questions, we can generate two types of hypotheses null hypothesis and alternative hypothesis:

- $\mathbf{H_0(1)}$: There is no difference in the term of response time and correctness when we use different style of curly brackets in program comprehension.

- $\mathbf{H_1(1)}$: There is a difference in the term of response time and correctness when we use different style of curly brackets in program comprehension.

- $\mathbf{H_0(2)}$: There is no difference in the term of visual attention when we use different style of curly brackets in program comprehension.

- $\mathbf{H_1(2)}$: There is a difference in the term of visual attention when we use different style of curly brackets in program comprehension.

3.1.4.1 Within Subject Design

After defining variables, research questions and hypothesis we are going to define choice of our study design which is within subject design [44]. In this design we make sure that each individual participant goes through all the condition, but it does not see the repeated code snippet even though it is in same code in different style. We can understand it with our study example. We have created three combinations for our study Combination 1, Combination2 , Combination 3. That we will see in Table 8.

We can see in each combination there are 8 snippets. The participant see snippet in the order same shows in the table. Algorithms are going to be same in order but for every combination there will be change in style of curly brackets used. Each snippet will be shown in two parts; First part will be calculation part and in second part participant will see the whole program with main function. We will see that in detail in next section with visual example.

Combination1	Combination2	Combination3
ArraySum(WOC)	ArraySum (WIC)	ArraySum(CNL)
BubbleSort(CNL)	BubbleSort(WOC)	BubbleSort(WIC)
NumberCheck(WIC)	NumberCheck(CNL)	NumberCheck(WOC)
ContainsSubstring(WOC)	ContainsSubstring (WIC)	ContainsSubstring(CNL)
Power(CNL)	Power(WOC)	Power(WIC)
BinaryConversion(WIC)	BinaryConversion(CNL)	BinaryConversion(WOC)
MultiplyMatrix(WOC)	MultiplyMatrix(WIC)	MultiplyMatrix(CNL)
PrimeNumber(CNL)	PrimeNumber(WOC)	PrimeNumber(WIC)

Table 8. Shows the combinations used to conduct the study

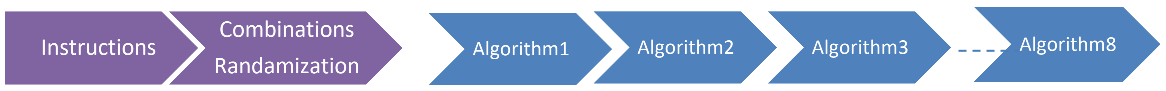


Figure 27. Visualization of our experiment design

We have described our experiment design in Figure 27. Where we can see first participant is going to see the instructions about the experiment where participant will understand about whole experiment process. Then one combination will be automatically selected for each individual participant's as shown in Table 8. We have generated the combinations in the format that each participant goes through all the possible conditions but do not see the same snippet again. As we can see in Table 8. This way we were able to differentiate between each participant and later we have grouped the participant according to combination they see.

3.3.5 Research setup

This section provides the details about tool and technology setup to conduct the experiment. How the experiment created with PsychoPy and how both Eye-Tracker and PsychoPy setup together to conduct the study.

a) **Setting up PsychoPy:** We have discussed PsychoPy in detail in section 3.3.3. We have used it in our study to generate the "questionnaire" for our participants.
Part 1 of the builder in our experiment is shown in Figure 28, and Part 2 is shown in Figure 29. We will explain each routine and loop in detail in this section.

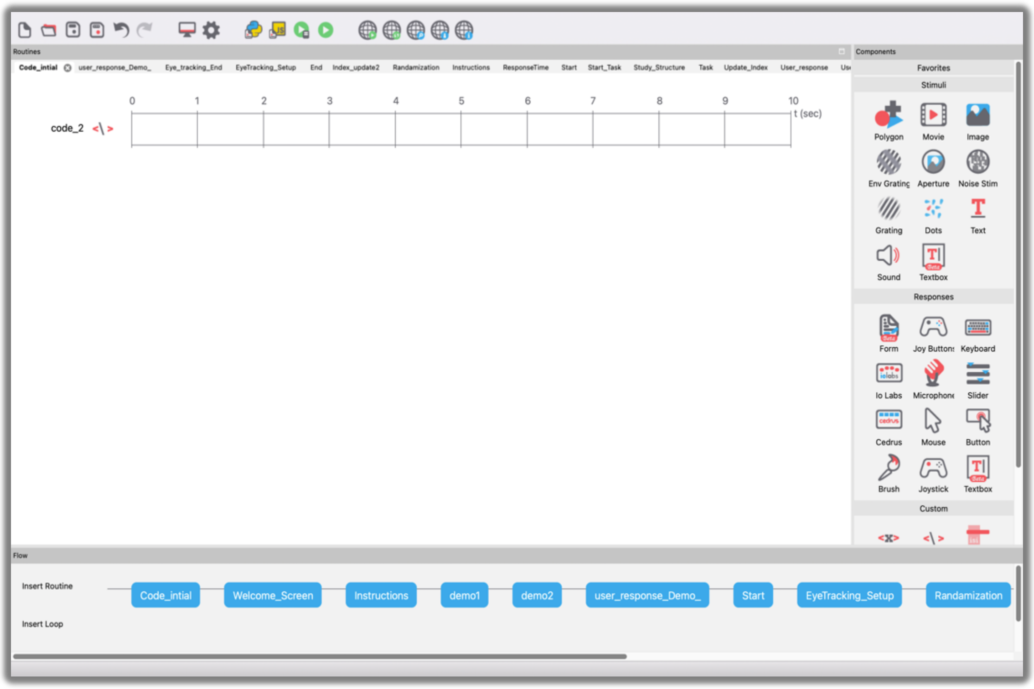


Figure 28. Builder with control flow of our experiment (Part1)

1. **Code initial:** This is the first routine of our experiment. In this routine we have written python script for importing our file and starting the experiment. We have used costume section of the builder which is provided by PsychoPy in its component section.

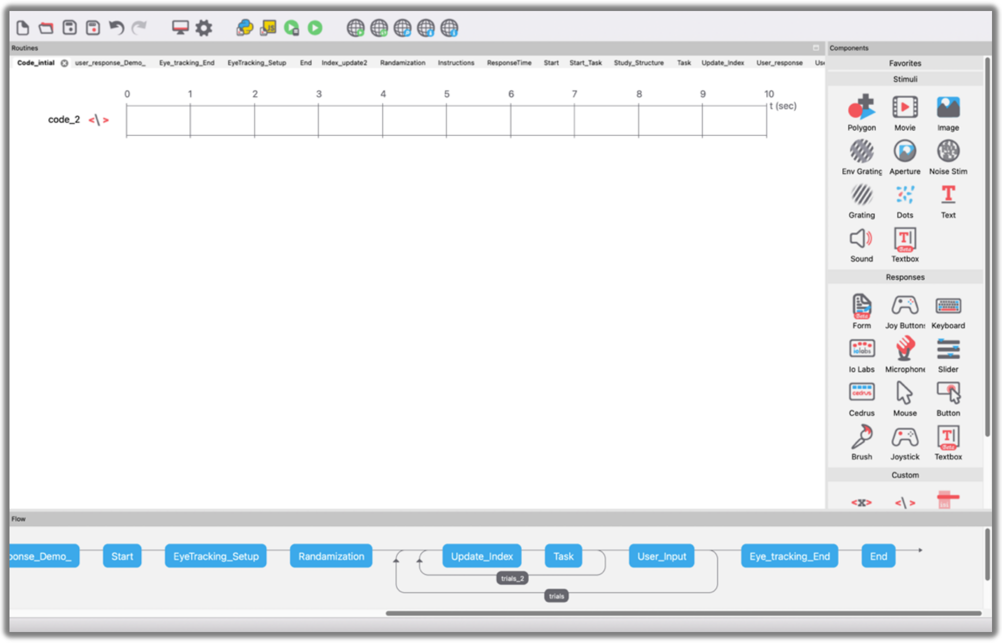


Figure 29. Builder with control flow of our experiment (Part2)

2. **Welcome Screen:** Welcome screen is a second routine where we have written welcome regards for our participant. We have used text stimuli from components section of builder to put the text on the study experiment.

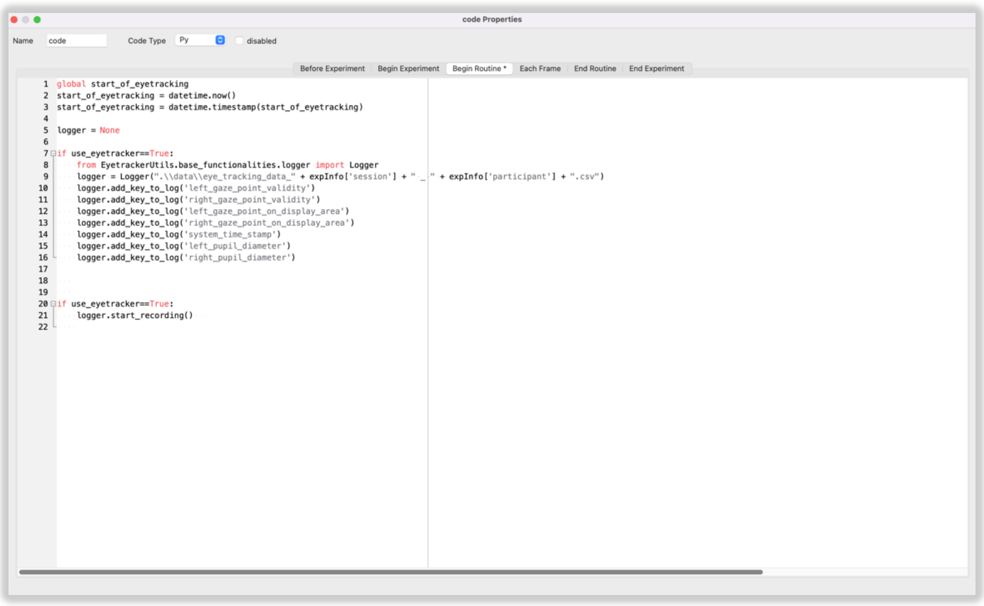
3. **Instructions:** In instruction section we have used again text stimuli to explain the experiment to the participant where he/she get to know about the process of the research and how to test is going to conducted.

4. **Demo1, Demo2:** In this routine there is a demo image for the participant before and experiment so they can get familiar with the experiment more. Demo1 contains an image with calculation part of snippet ad demo2 contains whole snippet with main function. This section is same for every participant. We have used image stimuli from the component to build this part.

5. **User response demo:** In this routine, we have created a demo response for the participant where they must choose from two options. To generate this part, we have used two responses from components. The mouse and the button**.**

6. **Start:** In this routine we have added a best wish for the student in the text form where we have added text stimuli from the components.

7. **Eye Tracking setup:** In this section we have synchronize the eye tracking device with the PsychoPy with the help of custom code section in components.



```
global start_of_eyetracking
start_of_eyetracking = datetime.now()
start_of_eyetracking = datetime.timestamp(start_of_eyetracking)

logger = None

if use_eyetracker==True:
    from EyetrackerUtils.base_functionalities.logger import Logger
    logger = Logger(".\\data\\eye_tracking_data_" + expInfo['session'] + "_" + expInfo['participant'] + ".csv")
    logger.add_key_to_log('left_gaze_point_validity')
    logger.add_key_to_log('right_gaze_point_validity')
    logger.add_key_to_log('left_gaze_point_on_display_area')
    logger.add_key_to_log('right_gaze_point_on_display_area')
    logger.add_key_to_log('system_time_stamp')
    logger.add_key_to_log('left_pupil_diameter')
    logger.add_key_to_log('right_pupil_diameter')



if use_eyetracker==True:
    logger.start_recording()

```

Figure 30. Shows eye tracking code setup with PsychoPy

8. **Randomization:** In this routine we can select a random combination for our participant. So, there is equal study for all the combinations.

9. **Loop(trail2):** This is the inner loop in flow diagram which contains two routines. First is update index which update the index of the upcoming stimuli and second is the stimuli we want show our participant. We created this loop because we have 16 images. First part is the calculation part, and second part is whole program, but we have only 8 algorithms. The inside loop runs those two images first and then goes to the participant response part.

10. **Loop(trail):** This loop is outer loop for running all 16 images. It runs 16 times. It contains that inner loop and one more stimulus that is user response or user input where user choose their answer.

11. **Eye tracking end:** This routine is outside both loops and in this we save the eye tracking data back to the file. So, we can use that data for analysis.

12. **End:** This routine is end part of the study where we thank the participant for their participation. We have used text stimuli in this.

b) **Setting up Eye tracker:** Eye tracker is an essential part of our study will give us data for required dependent variables which are response time, correctness, and most important visual attention. Therefore, we need to setup eye tracker in most accurate way. We have used table mounted Tobii eye tracker for our study with 120 Hz frequency. To setup table mounted eye tracker, we need the desktop monitor to mount on. We have used AOC monitor with the resolution of 1920*1080(width*height). Here are the steps to setup the eye tracker.

1. **Placing Eye tracker:** First step of the setting up eye tracker is fixing eye tracker with monitor. Eye tracker is fixed on the bottom of the monitor. Table mounted eye tracker provide us the point of regard for that it is important the position of eye tracker where we can get the perfect angle with the participant eyes.

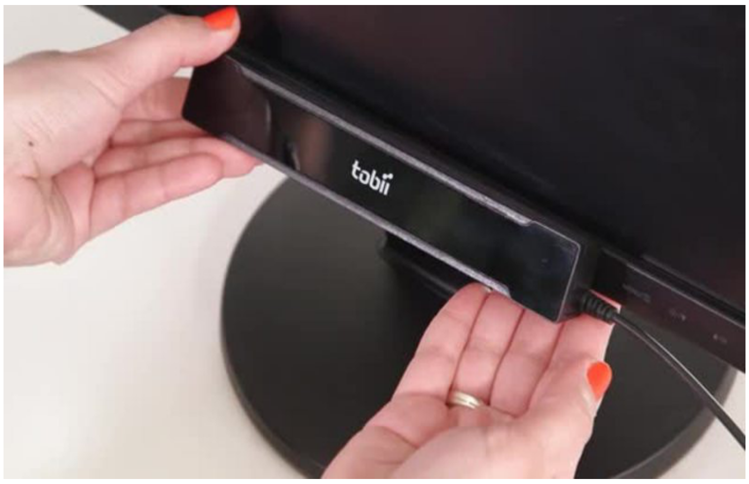


Figure 31. Placing an eye tracker

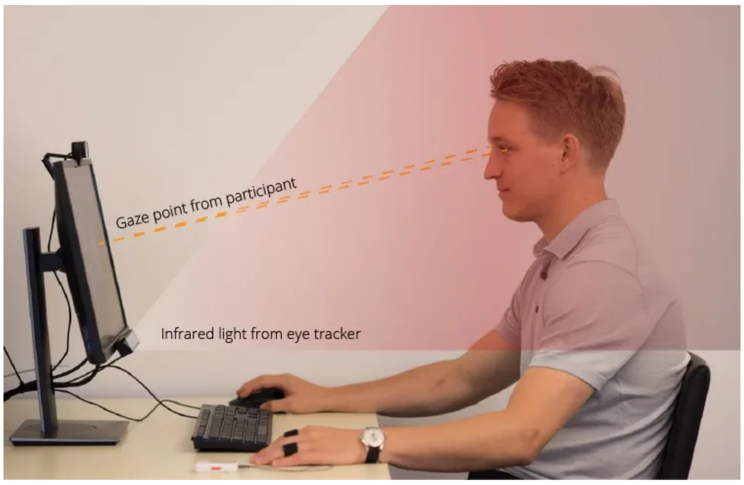


Figure 32. Participant's position for study

2. **Eye - Tracker Calibration:** After placing eye tracker participants was invited to do the test. First participant was told to sit on the chair and fix their head position according to eye tracker position. Tobi eye tracker gives the steps for calibration. First, we get the help from the screen to adjust our head position, according to the screen suggest.

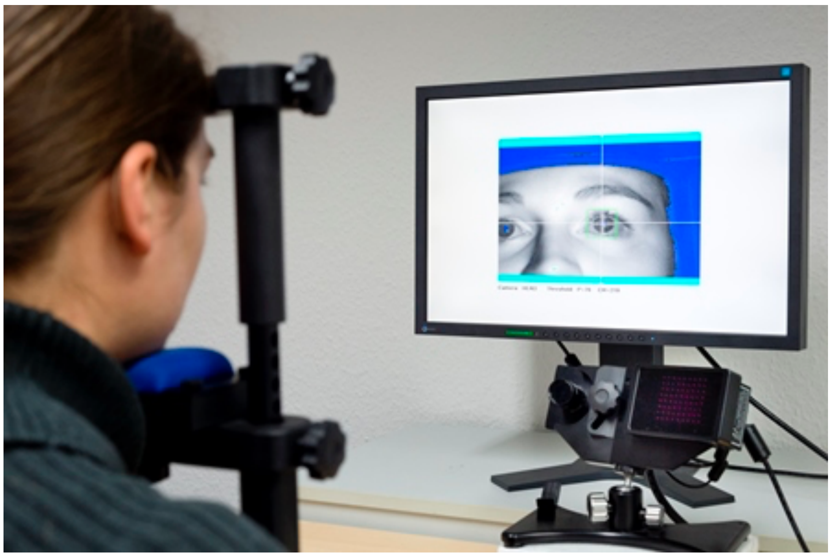


Figure 33. Adjusting head position with eye tracker

Screen shows us a square block where we need to fit our head so our eyes can be perfectly fit to screen. Second step is, user ask to look at specific points on the screen also known as calibration dots. During this period several images of eyes are collected and analyzed. The resulting information is then integrated in the eye model and the gaze point for each image sample is calculated.

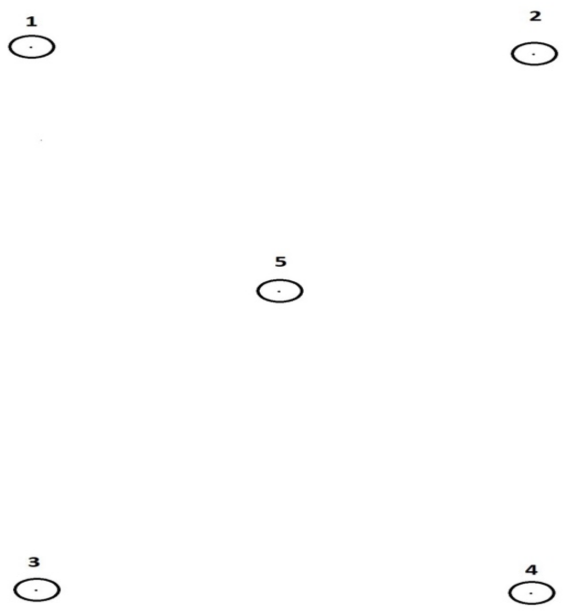


Figure 34. Calibration dots

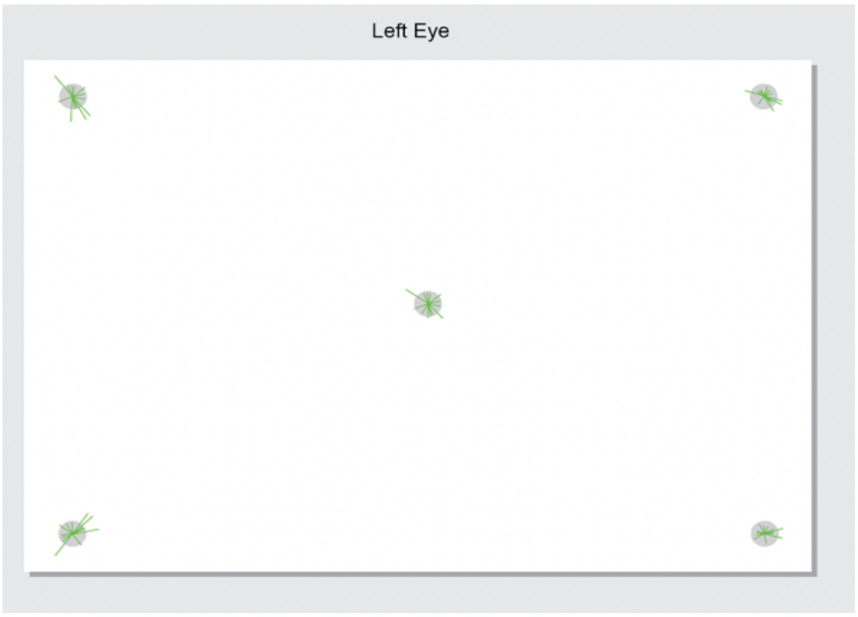


Figure 35. Calibration for left eye

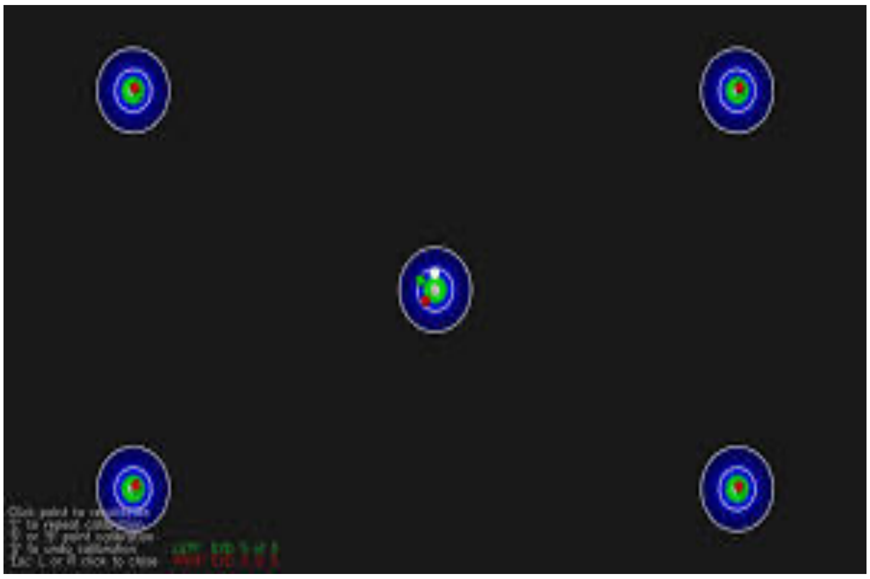
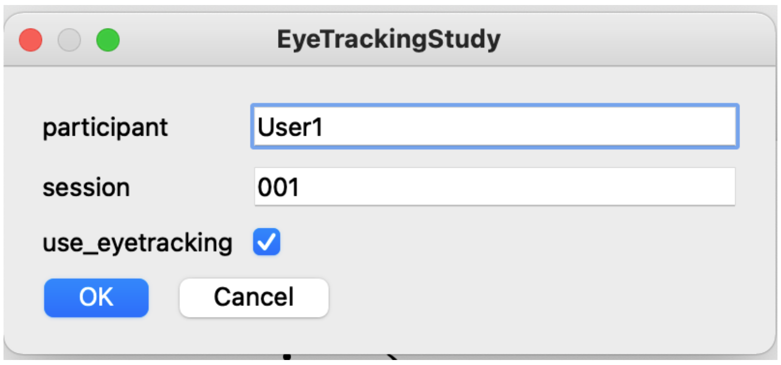


Figure 36. resultant image for calibration

In Figure 34 we can see the calibration dots eye tracker place those dots in all direction of the screen to get the gaze data more accurate. We can see the dots with numbering on it. This is the sequence eye tracker display those dots one by one. When the method is completed, the quality of the calibration is represented by green lines of varied length. We can see figure 34 the length of each line shows the offset between each sampled gaze point and the calibration dot's center. Large offsets (long green lines) may be caused by a variety of circumstances, including the user not concentrating on the point, being distracted during the calibration, or the eye tracker not being properly set up. In Figure 35 we see the calculation for left eye and the same calculation goes for the right eye. After that we can see the resultant image for calibration in figure 36 where we can see calibration points where left eye gaze points and right eye gaze points fall together in one circle. If points look together and not distracted from each other than the calibration seems good. However, if points are not seeming together and looks all over on the screen that means the calibration is not perfect in that case, we need to recalibrate the eye tracker.

c) **Final setup:** As we have seen earlier the PsychoPy is set up accurately and eye tracker is also set up accurately, in this step we start our study by pressing the run button on PsychoPy. After that participant need to fill the information about him/her name and session number. In this study we have told participant to not enter their actual name, instead of actual name we have given the name in the form User 1…. User n. We can see in figure 37 that participant has filled the information about them. Participant section is the participant's name that is User1 in this case, and session number is which combination they are using, we have kept the session number always 001 because we already know which participant have seen which combination.


EyeTrackingStudy

participant: User1

session: 001

use_eyetracking: ✓

Figure 37. Start of study

3.3.6 Data collection Method

After experiment is done data is saved automatically in the .csv file for each participant. We have discussed that while setting up PsychoPy. In the section end of Eye tracking where data will be saved back to the .csv file. We have collected the important parameters like 'left gaze point validity', 'right gaze point validity','left gaze point on display area', 'right gaze point on display area','system time stamp', 'left pupil diameter', 'right pupil diameter'. These are saved in a file with the column name. After that this .csv file is imported in Jupiter notebook with python script. After cleaning the data, important eye tracking matrixes like, fixation detection, saccade detection has been calculated. Later heatmaps, fixation points and scan path has been drowned on the snippets. In next chapter we are going to discuss how the results and data analysis has been performed.

4 Results and data analysis

Details on carrying out the experiment and gathering data are presented in the last section. Here, we'll go over data analysis and data cleaning.

4.1 Preparation of data set

The work of preparing the data set before beginning the analysis is rather crucial. We should get rid of any data that is erroneous or unnecessary if we want the findings to be accurate. In our technique of data collecting, which we mentioned in the chapter before this one, we have acquired data from the participants, and some of the data is not sufficiently accurate to be included in the data analysis. To exclude irrelevant information from the data, we have established a cutoff point, which states that an individual will be considered an outlier if they spend less than 30 seconds on the snippet and choose responses that are not correct.

During the process of data preparation, we found the data which done not fall into our criteria which are shown in Table 9. There are two columns in table participants and Snippets. In participant column we have participant names which are User15, User12 and User14 and in the next column we have corresponding snippets. We could not take those snippet's data into analysis because they fall into the condition of outliers.

Participants	Snippets
User15	BubbleSort (CNL), ConatinsSubstring (WOC)
User12	ArraySum (WOC)
User14	ArraySum (CNL)

Table 9. Shows all outliers corresponding participants and Snippets

4.2 Behavioral Data

RQ1(Response time and correctness): If the use of curly brackets in different styles makes any difference in the term of response time and correctness. After preprocessing of data, we are ready to go with further process. The data obtained from the survey is organized into the table, which will be analyzed to produce acceptable scientific conclusions.

The formulated data sets are organized into a table where we can see the multiple values of data sets. They are demonstrated in following Table 10:

	Without curly		With curly		Curly new line	
Snippets	**Response Time (ms)**	**Correctness (%)**	**Response Time (ms)**	Correctness (%)	**Response Time (ms)**	**Correctness (%)**
ArraySum	41 ± 22	100%	64 ± 29	83%	64 ± 14	100%
BubbleSort	91 ± 46	67%	115 ± 59	86%	60 ± 22	100%
NumberCheck	65 ± 34	86%	51 ± 18	100%	74 ± 36	67%
ContainsSubstring	75 ± 22	71%	91 ± 31	83%	92 ± 27	71%
Power	66 ± 26	33%	68 ± 26	57%	48 ± 26	88%
Binary Conversion	76 ± 29	86%	52 ± 28	88%	86 ± 28	67%
Matrix Multiplier	83 ± 57	67%	84 ± 37	50%	113 ± 55	29%
PrimeNumber	51 ± 28	67%	51 ± 28	43%	51 ± 28	75%
Total	**69± *33***	*72%*	***76± 34***	*74%*	**74± 30**	**75%**

Table 10. Complete list of research snippets, together with average response time (in seconds) and accuracy (in percent) across all three categories: Missing curly brackets, with curl brackets same line, with curly brackets new line.

4.2.1 Descriptive Statical analysis of behavioral data (Response time):

Table 10 summarizes the behavioral data, which is divided into three categories. The table shows the individual average for each snippet in all different format and shows overall average for all three formats.

We are going to discuss our **RQ1**: research question one in two parts in first part we are going to discuss how the response time is going to behave in all three categories (discussed in table 10). Second, we are going to evaluate the effect in term of correctness, how it makes difference in all three formats.

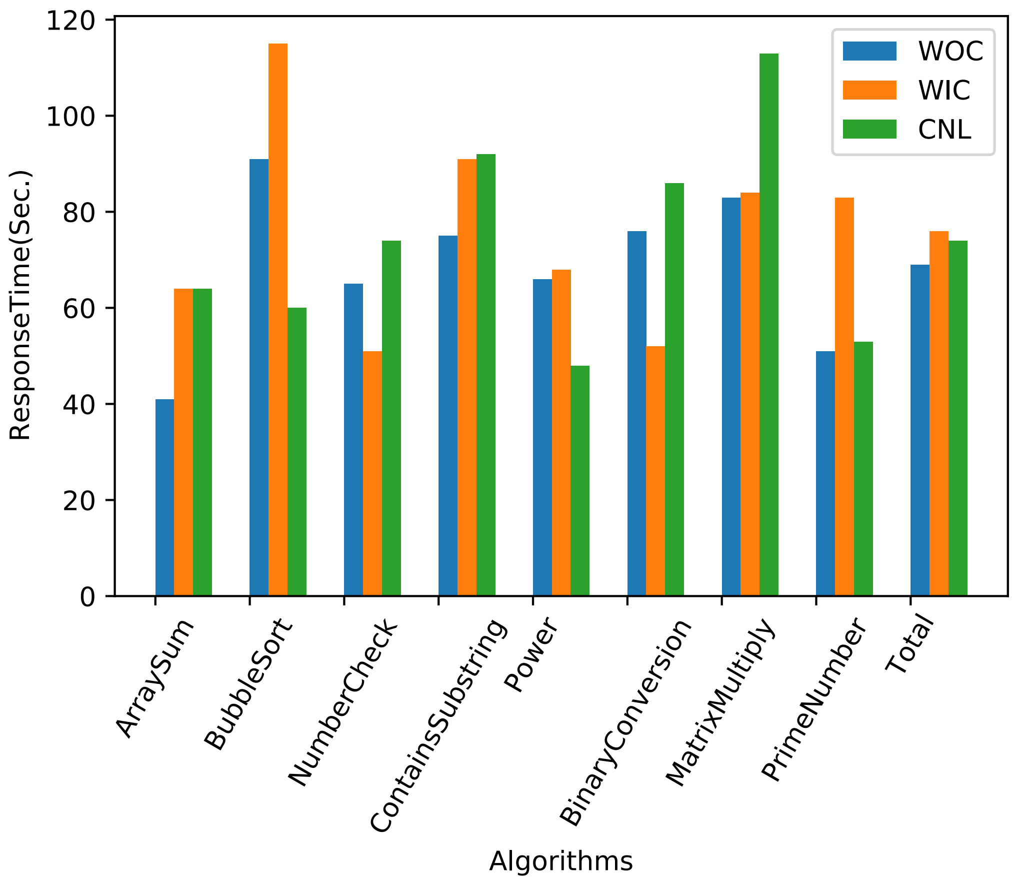


Figure 38. Demonstrates the mean response time for each participant in each format (Without curly brackets, With curly brackets from same line, with curly brackets from new line)

As we go in the details of the Table 10, First we are going to di comparison with total mean response time, and we can observe that total of mean response time for algorithms without curly brackets is the lowest and for with curly brackets is highest. If we compare mean response time individually, the table also gives us information that the BubbleSort (WIC) (in orange color), MultiplyMatrix (CNL)(in green color), BubbleSort (WOC)(in blue color) has the highest mean response time respectively (115, 113 and 91 seconds). On the other hand, ArraySum (WOC), Power (CNL), has minimum mean response time respectively (41,48 seconds).

After observing the Figure 38 In total mean response time, there is no such difference in all three formats but if consider without curly brackets for all format has lowest total mean response time and with curly brackets from same line has heights total mean response time.

4.2.2 Descriptive Statical analysis of behavioral data (Correctness):

As we go further in the descriptive analysis of behavioral data now, we go to the statics of correctness percentage of each participant for all three formats.

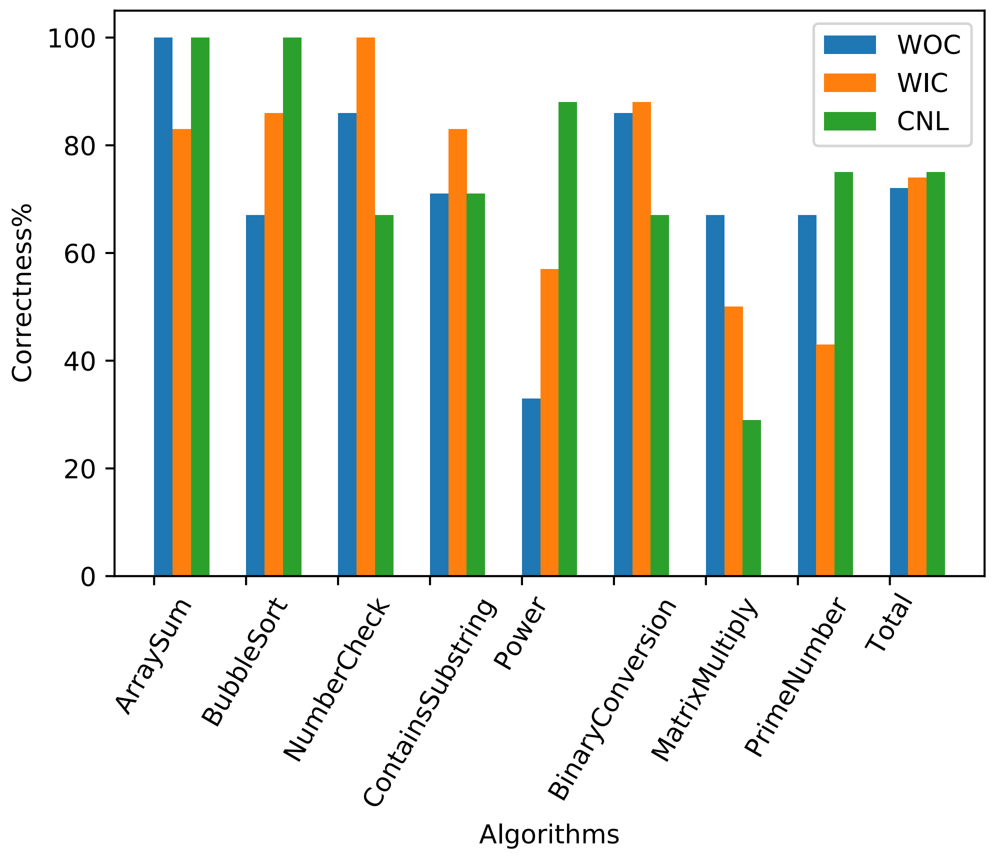


Figure 39. Demonstrates the average correctness percentage (%) for each participates in each snippet in all formats (Without curly brackets, With curly brackets from same line, with curly brackets from new line).

The second part of research question one (**RQ1**) is about the correctness of participants while doing the study. To evaluate the study, we have calculated correctness percentages for each participant and later evaluated the average percentage for each snippet in each format. We are going to observe how the correctness for each curly bracket style.

As we can observe in figure 39 the correctness percentage for without curly brackets stye is lowest and for curly bracket new line is highest, however there not a big different on total average correctness for each style of code which is respectably minimum to maximum (72%,74%,75%).

As we go more details in figure 39, we find that the individual correctness responses. ArraySum (WOC), ArraySum (CNL) NumberCheck (WIC), and BubbleSort (CNL) has accuracy of 100%. On the other hand, Power (WOC), MultiplyMatrix (CNL), PrimeNumber (WIC)shows the lowest accuracy respectively (33%,29%,43%).

4.2.3 Hypothesis Testing:

For this research statical analysis is essential to see weather formulated data can be tested to determine the significance value. The data in table 10 will be used to evaluate the hypothesis testing for research. As the table shows the mean response time and correctness (%) for each algorithm in each curly bracket style. We have already removed the outliers and we have already data to process.

4.2.4 Inferential Statical Analysis of Behavioral Data (Response Time):

As we have already evaluated the results for behavioral data in response time. In this section we are going to check the significance of the data. We are going to take 95% confidence interval and alpha value of 0.05. To evaluate the significancy of data we are going to use One-way ANOVA test with repeated measures. We are using ANOVA test for this because we have one independent variable which is programming styles with three categories (WOC, WIC, CNL), we could have used another test like t-Test if we have only two categories. We have divided our independent variable in three groups that is Algo (WOC), Algo (WIC), Algo (CNL). Algo (WOC) will contains all the mean values of response time for each algorithm, which are missing a curly bracket. Algo (WIC) is going to contain all the algorithms mean response time where curly brackets are starting from same line. At last, Algo (CNL) is going to contains the means of each algorithm where curly bracket is starting from new line.

First of all, we need to check the vital assumptions of ANOVA test which are normality of samples(residuals) and equality of variance. We have cheeked normality of samples with Q-Q plot and Shipro Wilk test. Firstly, we will explore Q-Q plot.

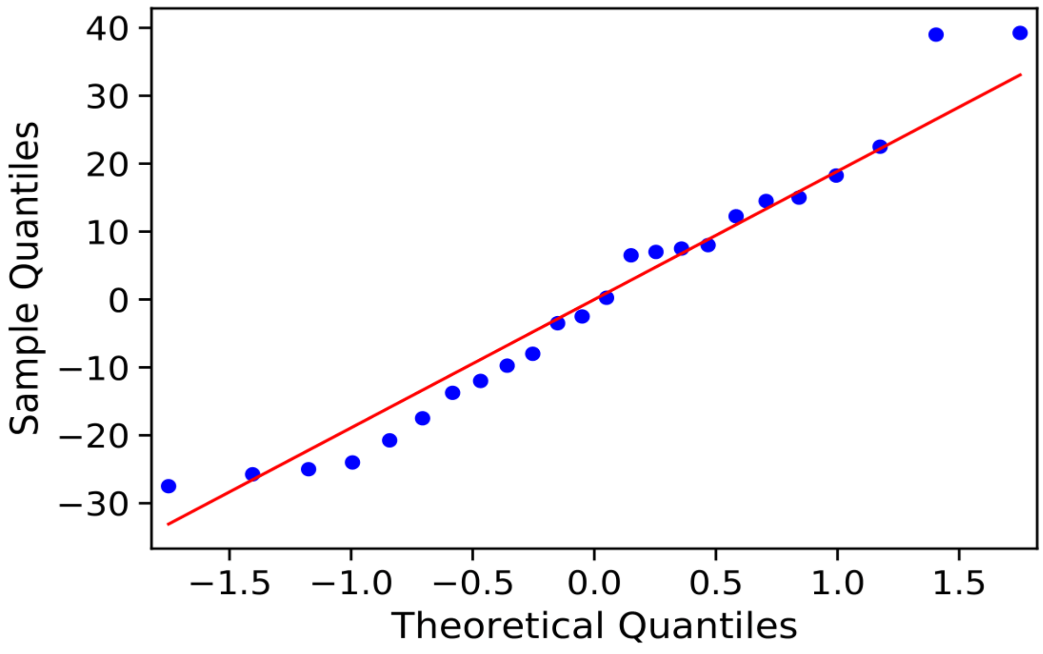


Figure 40. Q-Q plot to check the normality of residual

As we can see in Figure 40 that samples are near to the red line, and there is not much distortion on the samples. So, we can say that samples are approximately normally distributed and with the help of this information we have less chances of false positive results, which makes our significance result stronger. Shipro Wilk normality test provide us Pvalue = 0.301 which is higher than 0.05 so we can say samples are normally distributed. We have conducted LEVENE test of variance to check the variance of samples if they differ much or not, where we found Pvalue = 0.641, so we can say that there is not significant difference in variance of samples.

Furthermore, we can see in Figure 41 where we have generated the boxplot to see how the mean of each group varies with each other where it gives us the information that there is no significant difference between the mean of each group, as these overlaps with each other. To be a results significant boxes of mean should not overlap, should have enough gap between them.

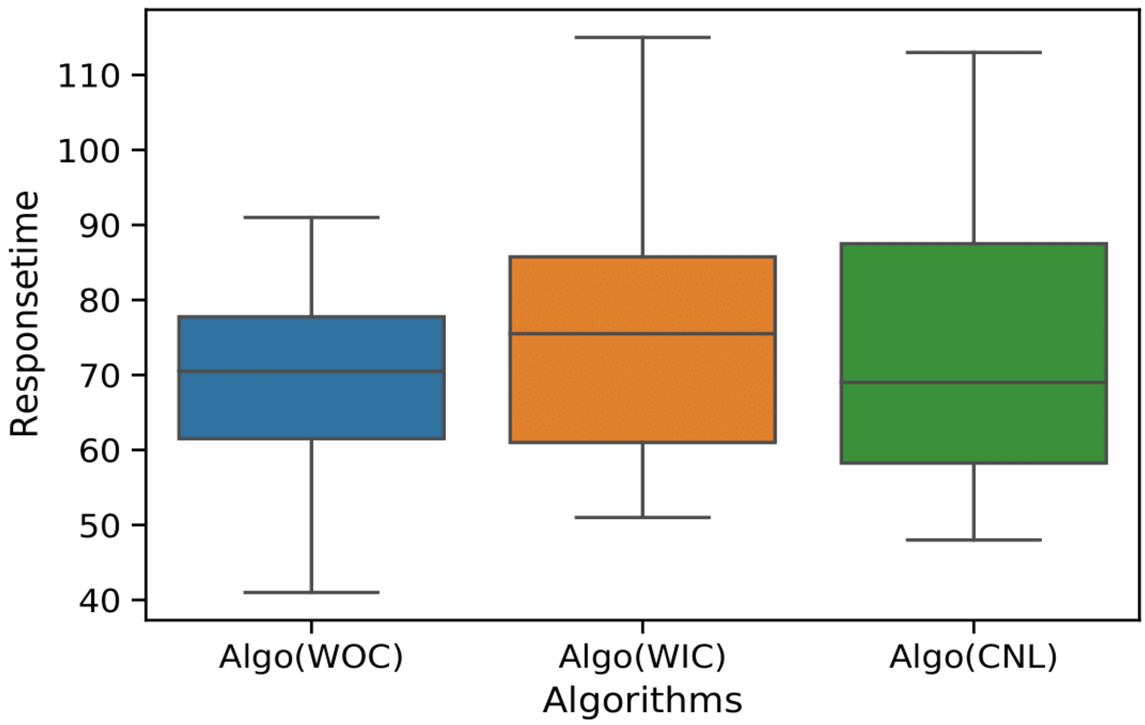


Figure 41. Boxplot comparing the response time mean for each group

Now we will apply ANOVA test and see the final results-

Main effect	Sum_sq	F value	PR(>F)	η²
Programming Style	237.0	0.2905	0.750	0.02
Residual	8565.5	-	-	-

Table 11. One-way ANOVA test results for response time

As we can clearly see in the table 11, (P-value = 0.750) which way bigger then α = 0.05. To be more sure we have also calculated effect size η^2=0.02, which way near to 0. So, we can say we have null effect size. I will help us to make our result stronger.

So, after analyzing the result of ANOVA test, we can come to conclusion that there is no significant evidence that the use of curly brackets in different style make any difference in the term of response time.

4.2.5 Inferential Statical analysis of behavioral data (Correctness (%)):

In this section we are going to discuss the correctness rate of different participant for different algorithms, and how much significance it has on use of different curly brackets styles. To start our analysis, we will separate our algorithms percentages in different groups same as during response time. There is going to be three groups (Algo (WOC), Algo (WIC), Algo (CNL). We have converted our data in this form because it is requirement of our model. We are going to conduct our study with one way ANOVA test because we have one independent variable with three categories.

To continue the analysis, we are going to take confidence interval of 95% and alpha value of 0.05. We are using ANOVA test because we have one dependent variable with three different categories. First, we need to check the vital assumptions of ANOVA test which are normality of samples and equality of variance. We have cheeked normality of samples with Q-Q plot and Shipro Wilk test. Firstly, we will explore the Q-Q plot.

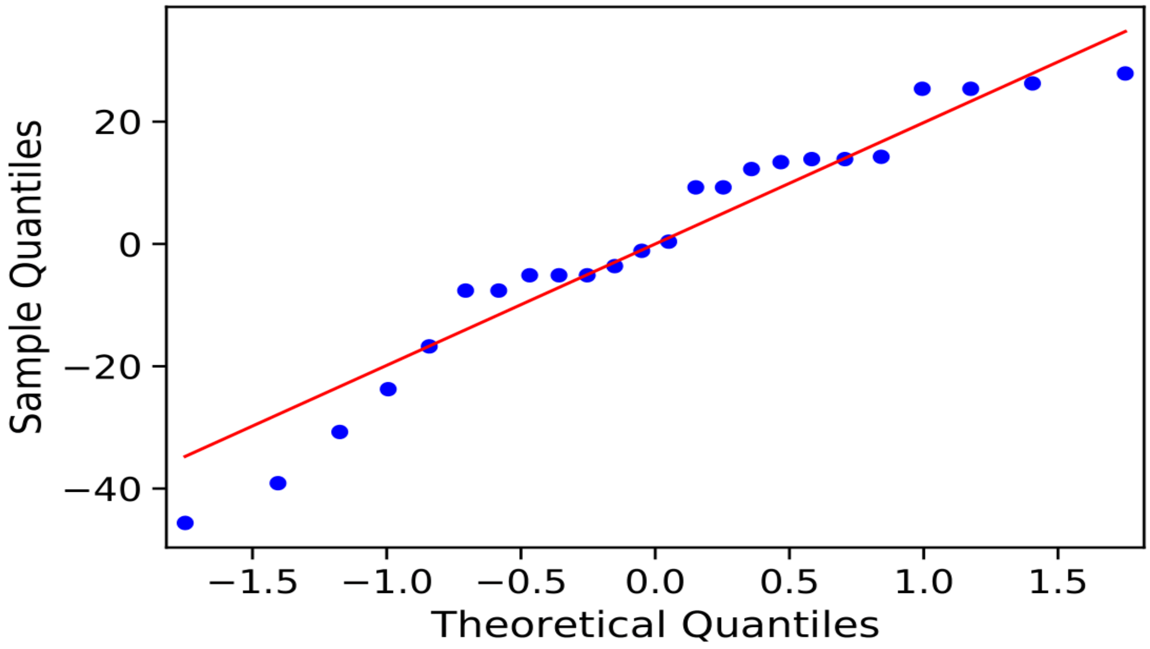


Figure 42. Q-Q plot to check the normality of residual for Correctness rate

As we can see in Figure 42 some samples are near to the red line, and some are, and some are far away which can lead us to false positive to solve this we will have conducted Shipro Wilk test provide us Pvalue = 0.301. So, we can say there is no violation of normality assumption and samples are normally distributed. To check the second assumption for the ANOVA test we have conducted LEVENE test to check the equality of variance where we found Pvalue = 0.942 which is far higher then 0.05

so there is no violation of second assumption and variance do not differ much from each other. Before applying ANOVA wee have also generated the box plot to check how the mean of each group varies. After visualizing the Boxplot in Figure 43, we can see that mean for each group are overlapping with each other and there is no significance difference between them. As, we have discussed earlier that to get the significant results Boxplot mean should not overlap

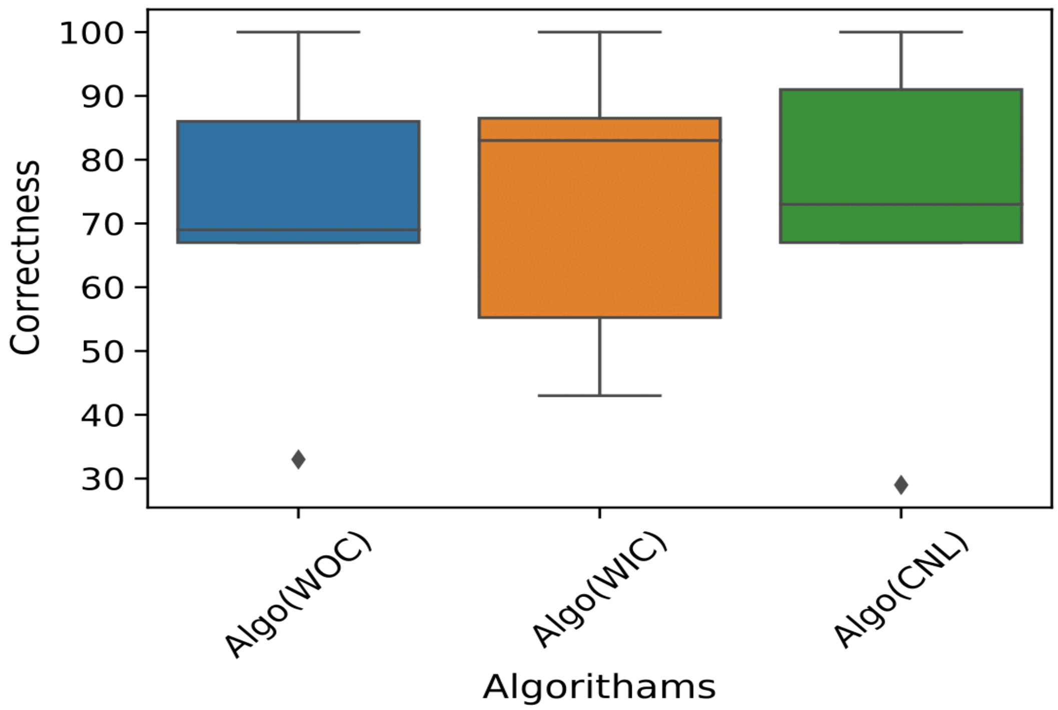


Figure 43. Boxplot comparing the response time mean for each group

Now we will apply ANOVA to see final results-

Main effect	**Sum_sq**	**F value**	**PR(>F)**	**η²**
Programming Style	25.75	0.028	0.970	0.002
Residual	8565.5	-	-	-

Table 12. One-way ANOVA test results for Correctness (%)

After observing the Table 12, we can say that as the (P-value = 0.975) which is nowhere near to our alpha value=0.05, and we also calculated effect size which is way near to 0 or null. So, we can say for now that there is no significance result that there is the in correctness (%) while using different style of curly brackets. So now we can come to conclusion for our first research question as we have results for both response time and correctness.

RQ1: Does the use of curly brackets in different style makes any difference in the term of response time and correctness?
Answer: There is no significant results which shows that there is difference in the term of response time and correctness while using different style of curly brackets.

4.3 Visual Attention:

In this section we are going to discuss about how the participants perform visually with data. We can get many answers from here from the visual patterns of participants. To perform this operation, we have used Tobii Pro Fusion Eye-Tracker tracker with 120hz frequency rate. After getting data from eye tracker with the help of PsychoPy we have generated 3 most important eye tracking matrixes.

1. Heatmaps
2. Fixation Points
3. Scanpath

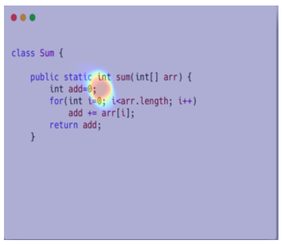
For our analysis purpose we are going to answer our second research question in this section which is **RQ2** – Does the use of curly brackets in different style makes any difference in the term of Visual attention?

To process the visual attention analysis first we are going to look at heatmaps. We can see in Table 13 where we have shown all the snippets with average heatmaps. We have categorized sippets in three group WOC (Without curly), WIC (with curly) and CNL (curly new lines). We have shown one snippet in two parts. First part is program without main function and second part where we can see the whole program. As we can see in heatmaps most of the focus is on main function and second focus is on calculation part. As we have chosen different type of snippets some are very simple, and some are medium complex. But none of the programmers are very complex because we are just observing that how programmer's eye moves during the comprehension.

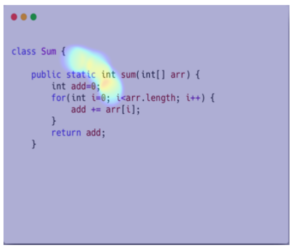
We have 6 images for each algorithm in Table 13 because we have generated heatmaps in both images. Those are when participant see only calculation part and after that participant see whole program.

WOC **WIC** **CNL**

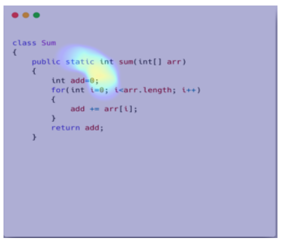
Array Sum



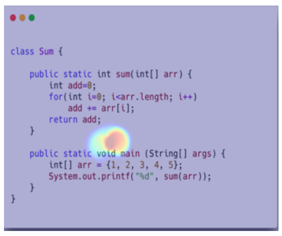
```
class Sum {

    public static int sum(int[] arr) {
        int add=0;
        for(int i=0; i<arr.length; i++)
            add += arr[i];
        return add;
    }
```



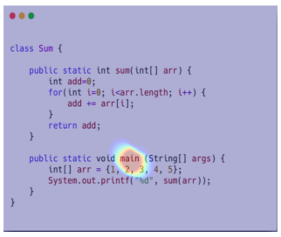
```
class Sum {

    public static int sum(int[] arr) {
        int add=0;
        for(int i=0; i<arr.length; i++) {
            add += arr[i];
        }
        return add;
    }
```



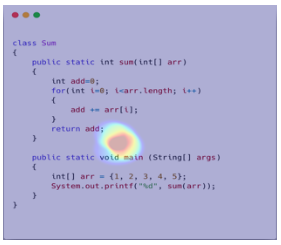
```
class Sum
{
    public static int sum(int[] arr)
    {
        int add=0;
        for(int i=0; i<arr.length; i++)
        {
            add += arr[i];
        }
        return add;
    }
```



```
class Sum {

    public static int sum(int[] arr) {
        int add=0;
        for(int i=0; i<arr.length; i++)
            add += arr[i];
        return add;
    }

    public static void main (String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

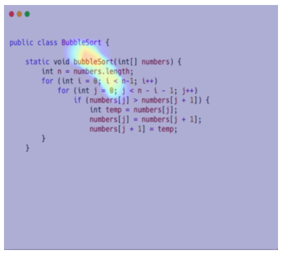


```
class Sum {

    public static int sum(int[] arr) {
        int add=0;
        for(int i=0; i<arr.length; i++) {
            add += arr[i];
        }
        return add;
    }

    public static void main (String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

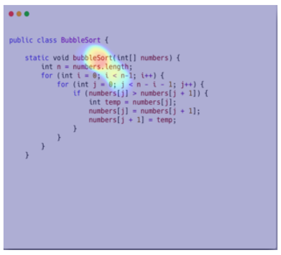


```
class Sum
{
    public static int sum(int[] arr)
    {
        int add=0;
        for(int i=0; i<arr.length; i++)
        {
            add += arr[i];
        }
        return add;
    }

    public static void main (String[] args)
    {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

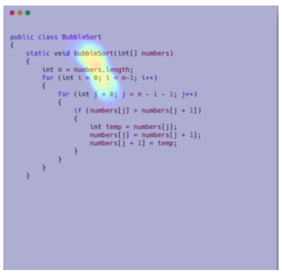
BubbleSort



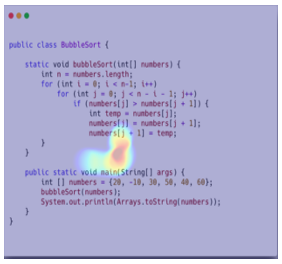
```
public class BubbleSort {

    static void bubbleSort(int[] numbers) {
        int n = numbers.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n - i - 1; j++)
                if (numbers[j] > numbers[j + 1]) {
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
    }
```



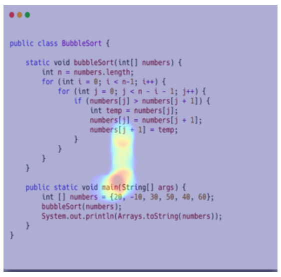
```
public class BubbleSort {

    static void bubbleSort(int[] numbers) {
        int n = numbers.length;
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (numbers[j] > numbers[j + 1]) {
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }
    }
```



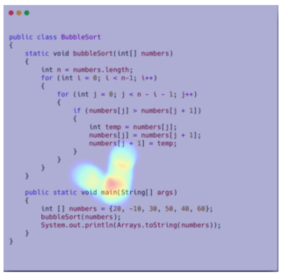
```
public class BubbleSort
{
    static void bubbleSort(int[] numbers)
    {
        int n = numbers.length;
        for (int i = 0; i < n-1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (numbers[j] > numbers[j + 1])
                {
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }
    }
```



```
public class BubbleSort {

    static void bubbleSort(int[] numbers) {
        int n = numbers.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n - i - 1; j++)
                if (numbers[j] > numbers[j + 1]) {
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
    }

    public static void main(String[] args) {
        int [] numbers = {20, -10, 30, 50, 40, 60};
        bubbleSort(numbers);
        System.out.println(Arrays.toString(numbers));
    }
}
```

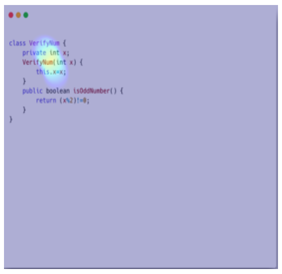


```
public class BubbleSort {

    static void bubbleSort(int[] numbers) {
        int n = numbers.length;
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (numbers[j] > numbers[j + 1]) {
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        int [] numbers = {20, -10, 30, 50, 40, 60};
        bubbleSort(numbers);
        System.out.println(Arrays.toString(numbers));
    }
}
```

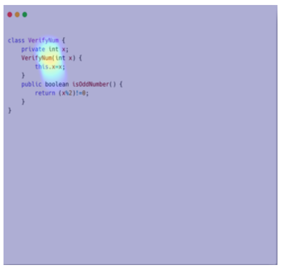


```
public class BubbleSort
{
    static void bubbleSort(int[] numbers)
    {
        int n = numbers.length;
        for (int i = 0; i < n-1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (numbers[j] > numbers[j + 1])
                {
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args)
    {
        int [] numbers = {20, -10, 30, 50, 40, 60};
        bubbleSort(numbers);
        System.out.println(Arrays.toString(numbers));
    }
}
```

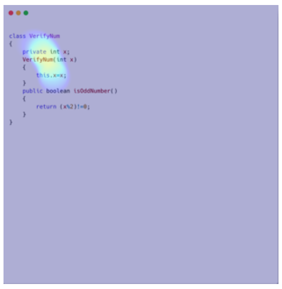
Number Check



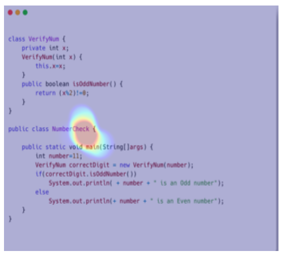
```
class VerifyNum {
    private int x;
    VerifyNum(int x) {
        this.x=x;
    }
    public boolean isOddNumber() {
        return (x%2)!=0;
    }
}
```



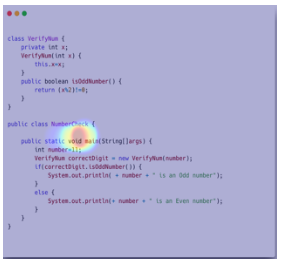
```
class VerifyNum {
    private int x;
    VerifyNum(int x) {
        this.x=x;
    }
    public boolean isOddNumber() {
        return (x%2)!=0;
    }
}
```



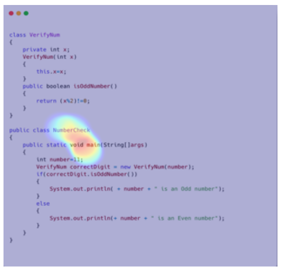
```
class VerifyNum
{
    private int x;
    VerifyNum(int x)
    {
        this.x=x;
    }
    public boolean isOddNumber()
    {
        return (x%2)!=0;
    }
}
```



```
class VerifyNum {
    private int x;
    VerifyNum(int x) {
        this.x=x;
    }
    public boolean isOddNumber() {
        return (x%2)!=0;
    }
}

public class NumberCheck {

    public static void main(String[]args) {
        int number=11;
        VerifyNum correctDigit = new VerifyNum(number);
        if(correctDigit.isOddNumber())
            System.out.println( + number + " is an Odd number");
        else
            System.out.println(+ number + " is an Even number");
    }
}
```

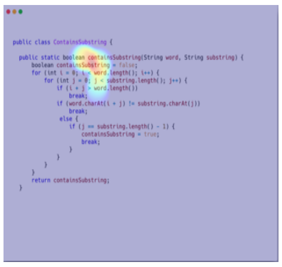


```
class VerifyNum {
    private int x;
    VerifyNum(int x) {
        this.x=x;
    }
    public boolean isOddNumber() {
        return (x%2)!=0;
    }
}

public class NumberCheck {

    public static void main(String[]args) {
        int number=11;
        VerifyNum correctDigit = new VerifyNum(number);
        if(correctDigit.isOddNumber()) {
            System.out.println( + number + " is an Odd number");
        }
        else {
            System.out.println(+ number + " is an Even number");
        }
    }
}
```

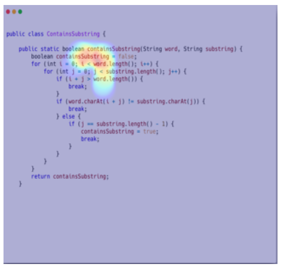


```
class VerifyNum
{
    private int x;
    VerifyNum(int x)
    {
        this.x=x;
    }
    public boolean isOddNumber()
    {
        return (x%2)!=0;
    }
}

public class NumberCheck
{
    public static void main(String[]args)
    {
        int number=11;
        VerifyNum correctDigit = new VerifyNum(number);
        if(correctDigit.isOddNumber())
        {
            System.out.println( + number + " is an Odd number");
        }
        else
        {
            System.out.println(+ number + " is an Even number");
        }
    }
}
```

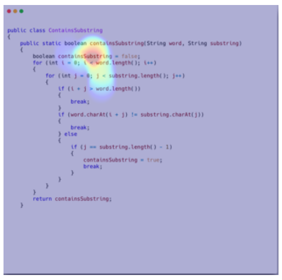
Contains Substring



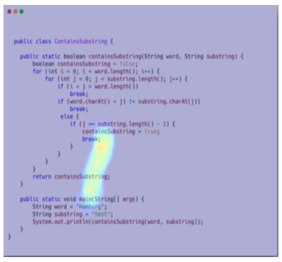
```
public class ContainsSubstring {

    public static boolean containsSubstring(String word, String substring) {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++) {
            for (int j = 0; j < substring.length(); j++) {
                if (i + j > word.length())
                    break;
                if (word.charAt(i + j) != substring.charAt(j))
                    break;
                 else {
                    if (j == substring.length() - 1) {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }
```



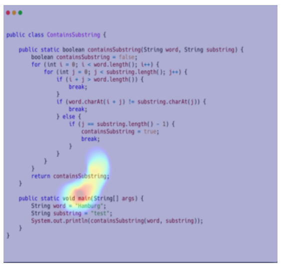
```
public class ContainsSubstring {

    public static boolean containsSubstring(String word, String substring) {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++) {
            for (int j = 0; j < substring.length(); j++) {
                if (i + j > word.length()) {
                    break;
                }
                if (word.charAt(i + j) != substring.charAt(j)) {
                    break;
                } else {
                    if (j == substring.length() - 1) {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }
```



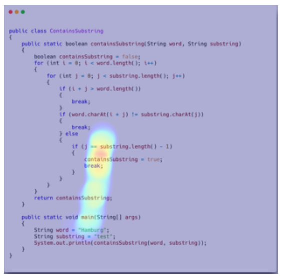
```
public class ContainsSubstring
{
    public static boolean containsSubstring(String word, String substring)
    {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++)
        {
            for (int j = 0; j < substring.length(); j++)
            {
                if (i + j > word.length())
                {
                    break;
                }
                if (word.charAt(i + j) != substring.charAt(j))
                {
                    break;
                } else
                {
                    if (j == substring.length() - 1)
                    {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }
```



```
public class ContainsSubstring {

    public static boolean containsSubstring(String word, String substring) {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++) {
            for (int j = 0; j < substring.length(); j++) {
                if (i + j > word.length())
                    break;
                if (word.charAt(i + j) != substring.charAt(j))
                    break;
                 else {
                    if (j == substring.length() - 1) {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }

    public static void main(String[] args) {
        String word = "Hamburg";
        String substring = "test";
        System.out.println(containsSubstring(word, substring));
    }
}
```

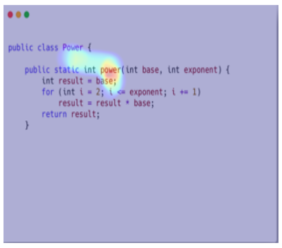


```
public class ContainsSubstring {

    public static boolean containsSubstring(String word, String substring) {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++) {
            for (int j = 0; j < substring.length(); j++) {
                if (i + j > word.length()) {
                    break;
                }
                if (word.charAt(i + j) != substring.charAt(j)) {
                    break;
                } else {
                    if (j == substring.length() - 1) {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }

    public static void main(String[] args) {
        String word = "Hamburg";
        String substring = "test";
        System.out.println(containsSubstring(word, substring));
    }
}
```

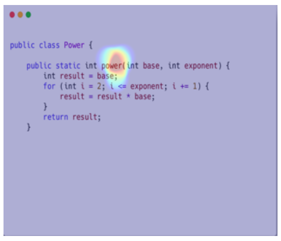


```
public class ContainsSubstring
{
    public static boolean containsSubstring(String word, String substring)
    {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++)
        {
            for (int j = 0; j < substring.length(); j++)
            {
                if (i + j > word.length())
                {
                    break;
                }
                if (word.charAt(i + j) != substring.charAt(j))
                {
                    break;
                } else
                {
                    if (j == substring.length() - 1)
                    {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }

    public static void main(String[] args)
    {
        String word = "Hamburg";
        String substring = "test";
        System.out.println(containsSubstring(word, substring));
    }
}
```

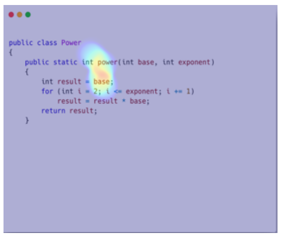
Power



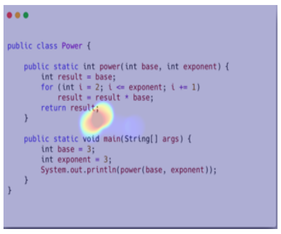
```
public class Power {

    public static int power(int base, int exponent) {
        int result = base;
        for (int i = 2; i <= exponent; i += 1)
            result = result * base;
        return result;
    }
```



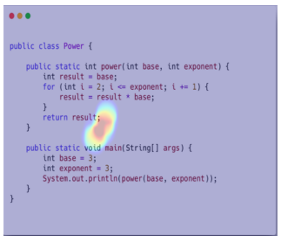
```
public class Power {

    public static int power(int base, int exponent) {
        int result = base;
        for (int i = 2; i <= exponent; i += 1) {
            result = result * base;
        }
        return result;
    }
```



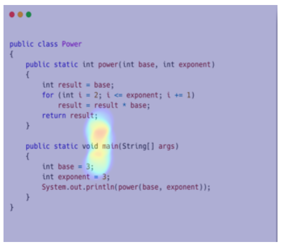
```
public class Power
{
    public static int power(int base, int exponent)
    {
        int result = base;
        for (int i = 2; i <= exponent; i += 1)
            result = result * base;
        return result;
    }
```



```
public class Power {

    public static int power(int base, int exponent) {
        int result = base;
        for (int i = 2; i <= exponent; i += 1)
            result = result * base;
        return result;
    }

    public static void main(String[] args) {
        int base = 3;
        int exponent = 3;
        System.out.println(power(base, exponent));
    }
}
```

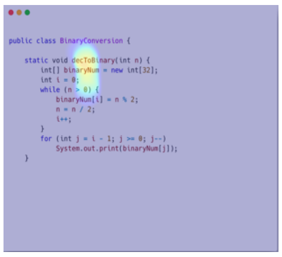


```
public class Power {

    public static int power(int base, int exponent) {
        int result = base;
        for (int i = 2; i <= exponent; i += 1) {
            result = result * base;
        }
        return result;
    }

    public static void main(String[] args) {
        int base = 3;
        int exponent = 3;
        System.out.println(power(base, exponent));
    }
}
```

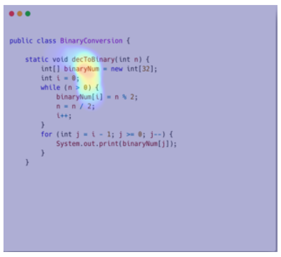


```
public class Power
{
    public static int power(int base, int exponent)
    {
        int result = base;
        for (int i = 2; i <= exponent; i += 1)
            result = result * base;
        return result;
    }

    public static void main(String[] args)
    {
        int base = 3;
        int exponent = 3;
        System.out.println(power(base, exponent));
    }
}
```

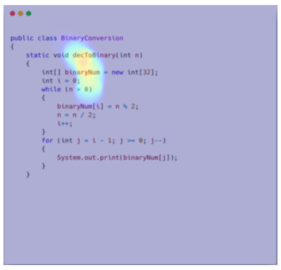
Binary Conversion



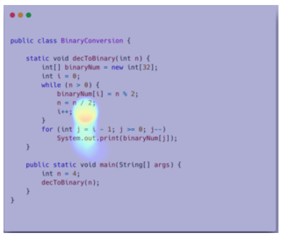
```
public class BinaryConversion {

    static void decToBinary(int n) {
        int[] binaryNum = new int[32];
        int i = 0;
        while (n > 0) {
            binaryNum[i] = n % 2;
            n = n / 2;
            i++;
        }
        for (int j = i - 1; j >= 0; j--)
            System.out.print(binaryNum[j]);
    }
```



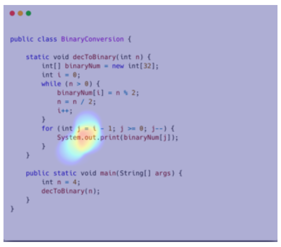
```
public class BinaryConversion {

    static void decToBinary(int n) {
        int[] binaryNum = new int[32];
        int i = 0;
        while (n > 0) {
            binaryNum[i] = n % 2;
            n = n / 2;
            i++;
        }
        for (int j = i - 1; j >= 0; j--) {
            System.out.print(binaryNum[j]);
        }
    }
```



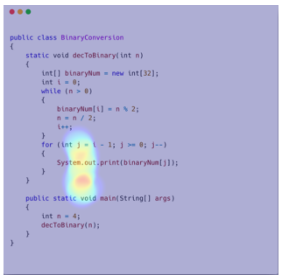
```
public class BinaryConversion
{
    static void decToBinary(int n)
    {
        int[] binaryNum = new int[32];
        int i = 0;
        while (n > 0)
        {
            binaryNum[i] = n % 2;
            n = n / 2;
            i++;
        }
        for (int j = i - 1; j >= 0; j--)
        {
            System.out.print(binaryNum[j]);
        }
    }
```



```
public class BinaryConversion {

    static void decToBinary(int n) {
        int[] binaryNum = new int[32];
        int i = 0;
        while (n > 0) {
            binaryNum[i] = n % 2;
            n = n / 2;
            i++;
        }
        for (int j = i - 1; j >= 0; j--)
            System.out.print(binaryNum[j]);
    }

    public static void main(String[] args) {
        int n = 4;
        decToBinary(n);
    }
}
```



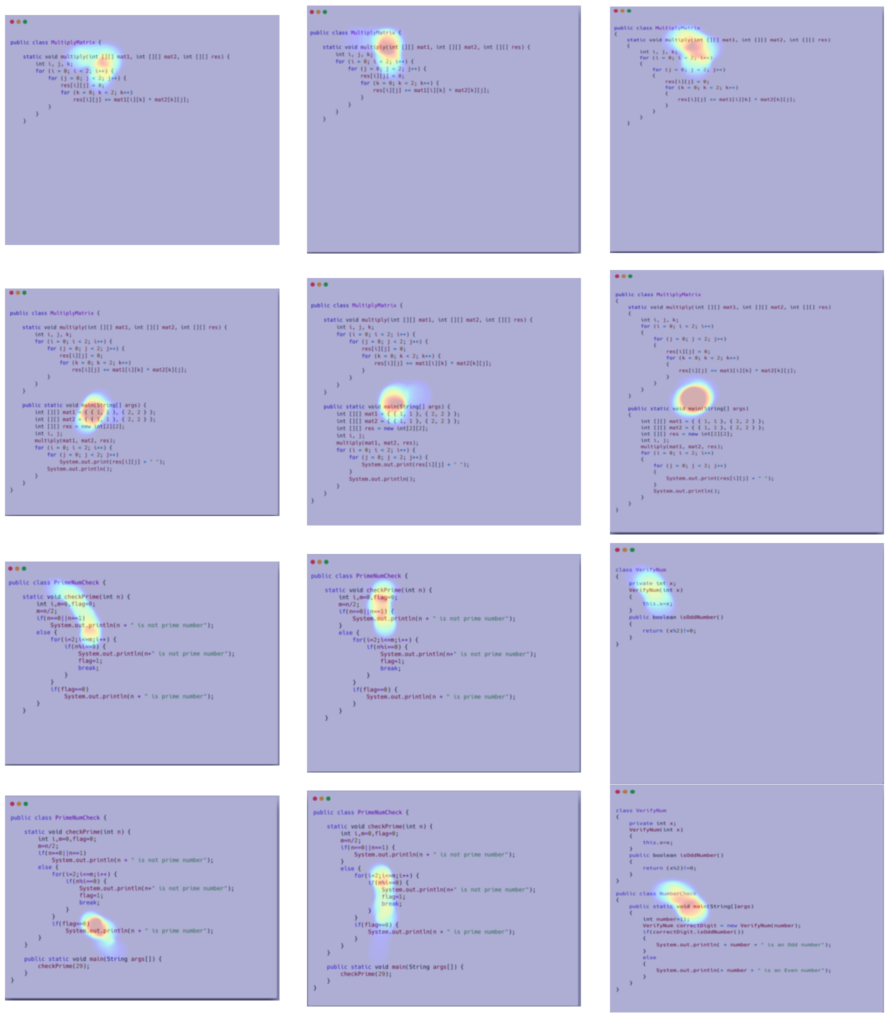
```
public class BinaryConversion {

    static void decToBinary(int n) {
        int[] binaryNum = new int[32];
        int i = 0;
        while (n > 0) {
            binaryNum[i] = n % 2;
            n = n / 2;
            i++;
        }
        for (int j = i - 1; j >= 0; j--) {
            System.out.print(binaryNum[j]);
        }
    }

    public static void main(String[] args) {
        int n = 4;
        decToBinary(n);
    }
}
```



```
public class BinaryConversion
{
    static void decToBinary(int n)
    {
        int[] binaryNum = new int[32];
        int i = 0;
        while (n > 0)
        {
            binaryNum[i] = n % 2;
            n = n / 2;
            i++;
        }
        for (int j = i - 1; j >= 0; j--)
        {
            System.out.print(binaryNum[j]);
        }
    }

    public static void main(String[] args)
    {
        int n = 4;
        decToBinary(n);
    }
}
```

Matrix Multiply

Prime Number



```
public class MultiplyMatrix {

    static void multiply(int [][] mat1, int [][] mat2, int [][] res) {
        int i, j, k;
        for (i = 0; i < 2; i++) {
            for (j = 0; j < 2; j++) {
                res[i][j] = 0;
                for (k = 0; k < 2; k++)
                    res[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }

    public static void main(String[] args) {
        int [][] mat1 = { { 1, 1 }, { 2, 2 } };
        int [][] mat2 = { { 1, 1 }, { 2, 2 } };
        int [][] res = new int[2][2];
        int i, j;
        multiply(mat1, mat2, res);
        for (i = 0; i < 2; i++) {
            for (j = 0; j < 2; j++)
                System.out.print(res[i][j] + " ");
            System.out.println();
        }
    }
}
```

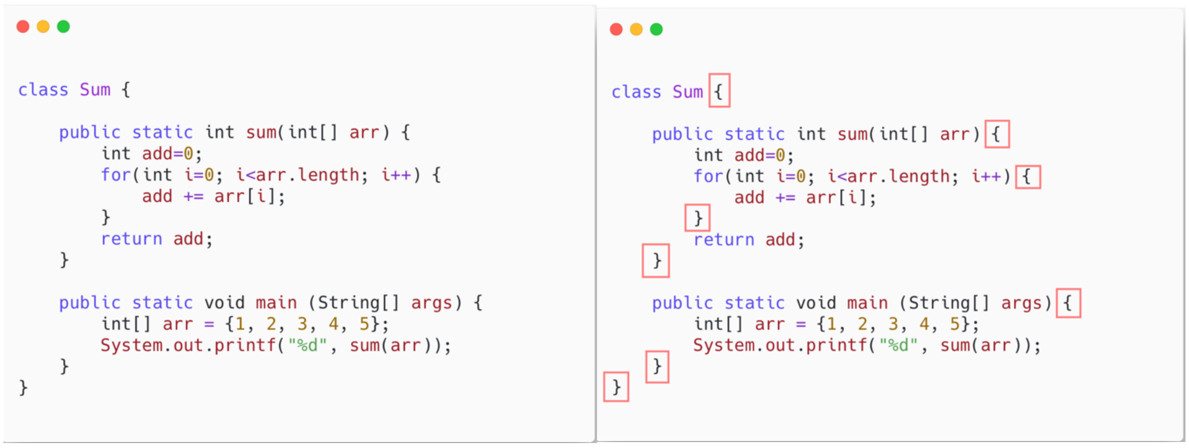
```
public class MultiplyMatrix
{
    static void multiply(int [][] mat1, int [][] mat2, int [][] res)
    {
        int i, j, k;
        for (i = 0; i < 2; i++)
        {
            for (j = 0; j < 2; j++)
            {
                res[i][j] = 0;
                for (k = 0; k < 2; k++)
                {
                    res[i][j] += mat1[i][k] * mat2[k][j];
                }
            }
        }
    }

    public static void main(String[] args)
    {
        int [][] mat1 = { { 1, 1 }, { 2, 2 } };
        int [][] mat2 = { { 1, 1 }, { 2, 2 } };
        int [][] res = new int[2][2];
        int i, j;
        multiply(mat1, mat2, res);
        for (i = 0; i < 2; i++)
        {
            for (j = 0; j < 2; j++)
            {
                System.out.print(res[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

```
public class PrimeNumCheck {

    static void checkPrime(int n) {
        int i,m=0,flag=0;
        m=n/2;
        if(n==0||n==1)
            System.out.println(n + " is not prime number");
        else {
            for(i=2;i<=m;i++) {
                if(n%i==0) {
                    System.out.println(n+" is not prime number");
                    flag=1;
                    break;
                }
            }
            if(flag==0)
                System.out.println(n + " is prime number");
        }
    }

    public static void main(String args[]) {
        checkPrime(29);
    }
}
```

```
class VerifyNum
{
    private int x;
    VerifyNum(int x)
    {
        this.x=x;
    }
    public boolean isOddNumber()
    {
        return (x%2)!=0;
    }
}

public class NumberCheck
{
    public static void main(String[]args)
    {
        int number=11;
        VerifyNum correctDigit = new VerifyNum(number);
        if(correctDigit.isOddNumber())
        {
            System.out.println( + number + " is an Odd number");
        }
        else
        {
            System.out.println(+ number + " is an Even number");
        }
    }
}
```

Table 13. Show the average heatmaps for all the combination

As we go further in analysis part, we have generated Area of Interest (AOI) for each image. AOI’s in our study are curly brackets. AOIs are the part of snippet where the participates supposed to focus on the study. These are (x, y) coordinates in image and we can say calculate the quantitative eye data.



```
class Sum {

    public static int sum(int[] arr) {
        int add=0;
        for(int i=0; i<arr.length; i++) {
            add += arr[i];
        }
        return add;
    }

    public static void main (String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

(a)

```
class Sum {

    public static int sum(int[] arr) {
        int add=0;
        for(int i=0; i<arr.length; i++) {
            add += arr[i];
        }
        return add;
    }

    public static void main (String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

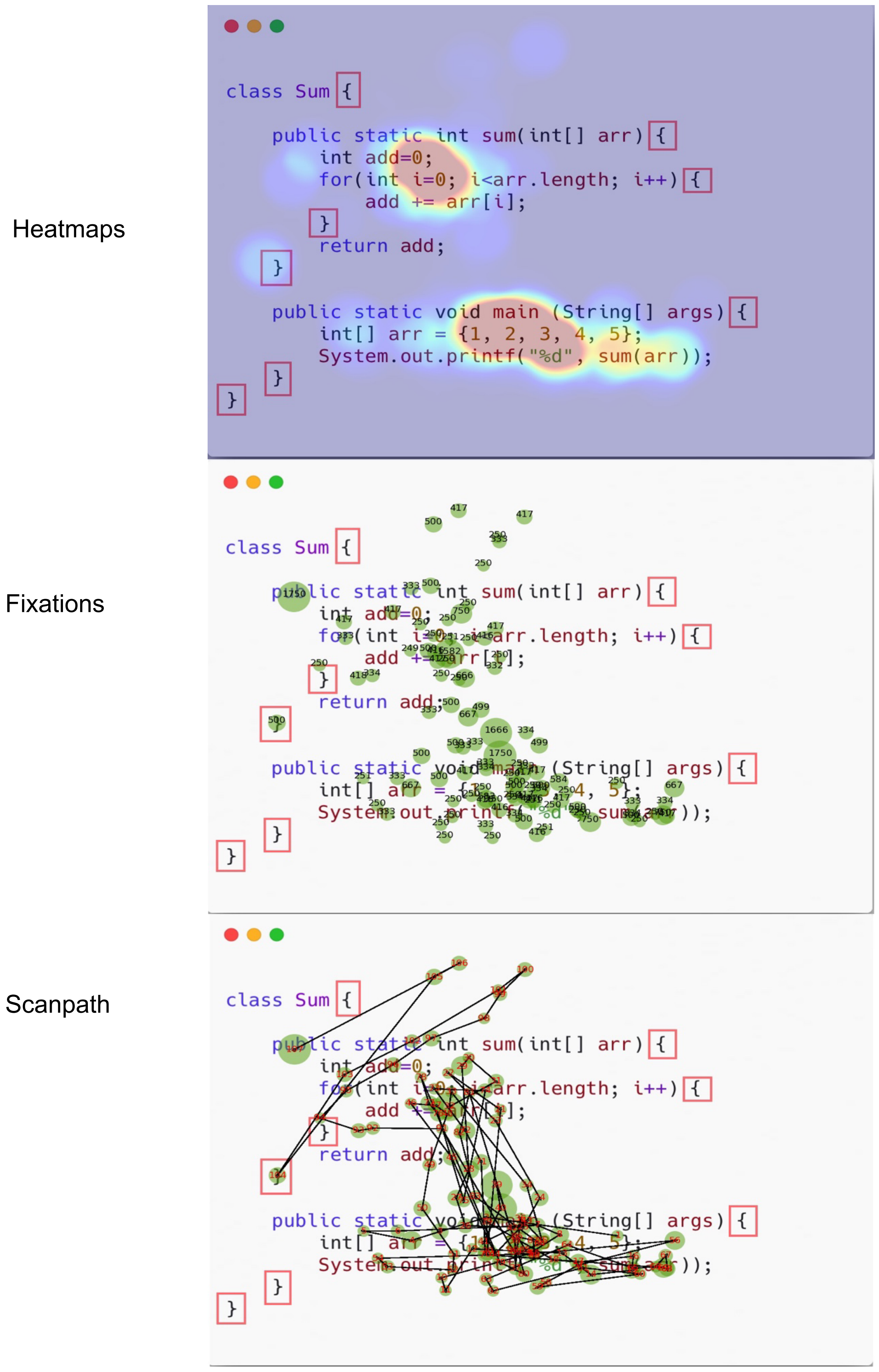
(b)

Figure 44. Shows the both the images together (a) ArraySum (without AOI’s) and (b) ArraySum (with AOI’s)

We can clearly see in Figure 44 that (a) tells how the snippet looks without AOIs and (b) tells us how snippet looks with AOIs. Now with AOIs it will be easy for us to know did the participant focuses on AOIs or not.

We have concentrated on other matrices as well, such as fixation and scanpath. These visual matrices make it easier for us to understand how users interpret the code. Fixation points let us determine where and how long a participant has been paying attention. Particularly when combined with AOIs, it becomes stronger and provides a larger perspective for us to comprehend. The second is the scanpath, which instructs us to utilize the participant's pattern to understand the code. We know the participant's eye movement path and assign numbers to it to make it simple to see.

The fixation points, scanpath, and heatmaps are all easily visible in figure 45. So that we can state that the user focused on the AOI for this specific picture, we can see that the 5th AOI comprises one fixation point with a value of 500 milliseconds. This information helps us to identify the focus point on the image.



```
class Sum {

    public static int sum(int[] arr) {
        int add=0;
        for(int i=0; i<arr.length; i++) {
            add += arr[i];
        }
        return add;
    }

    public static void main (String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

Figure 45. shows the heatmaps, fixation points and scnapath over AOIs of the Images

After visualizing the data now, we need to analyze that does visual attention makes really any difference regarding to the use of different style of curly brackets? For this purpose, we have calculated the average fixation duration for each AOI for each algorithm and then after that overall average for each style of brackets. We can see in table 14.

Algorithmus	**Without curly (Fixation Duration) (ms)**	**With Curly (Fixation Duration) (ms)**	**Curly newline (Fixation Duration) (ms)**
ArraySum	14	168	42
BubbleSort	350	152	116
NumberCheck	25	278	132
ContainsSubstring	228	778	1085
Power	166	235	216
BinaryConversion	187	29	1067
MultiplyMatrix	48	268	1500
PrimeNumber	199	445	92
Total	152	294	531

Table 14. Shows average fixation duration in AOI for each snippet

4.3.1 Descriptive Statical analysis of visual attention (Fixation Duration):

In this section we are going to see the describe effect of using different style of curly brackets for the visual attention by comparing the fixation duration. We have calculated average fixation duration for each snippet, and we are going to describe it with the help of bar chart where we can compare all the fixation duration of each snippet for different style of curly brackets used for analysis. We have converted time from second to milliseconds because in seconds we did not have enough big values to analyze, otherwise we could not compare it in better way.

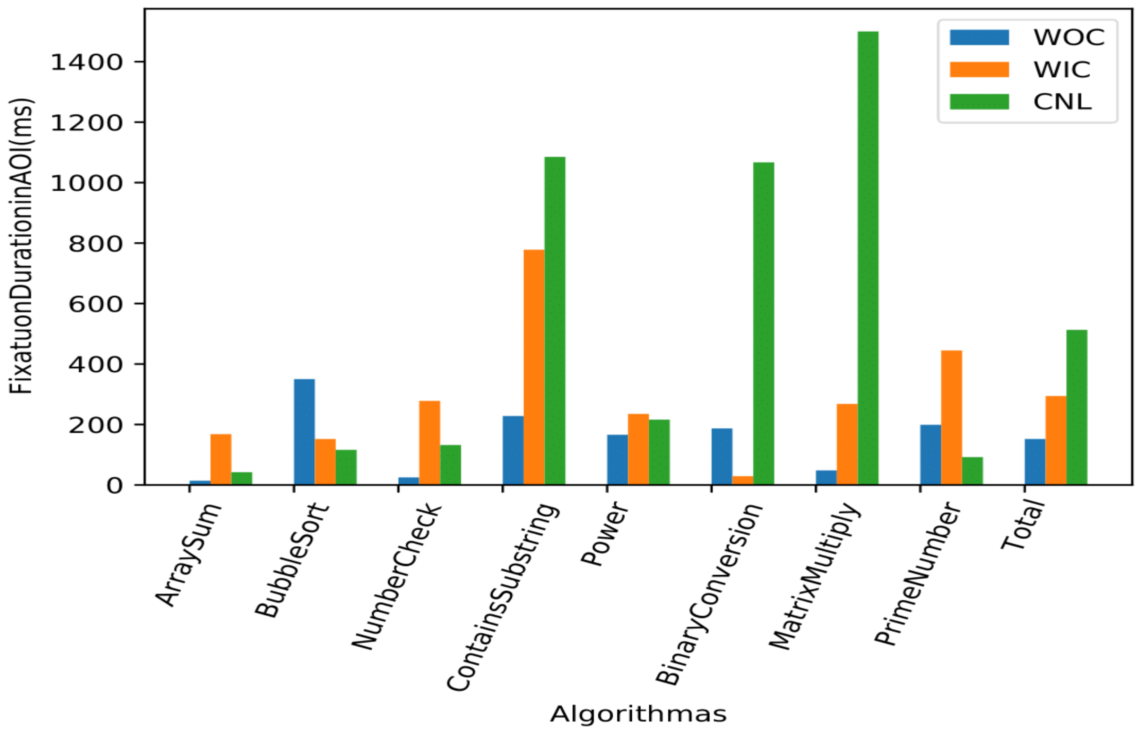


Figure 46. Shows the different response time for algorithms in different style of brackets

Figure 46 reveals that there are significant variations across the various methods, as can be seen here. MultiplyMatrix (CNL), ContainsSubstring (CNL), and Binary-conversion (CNL) are the primary three algorithms that stand out in terms of fixation duration. As is evident to us, all these algorithms were derived from the third type of curly brackets, which is referred as curly newline bracket. For the time being, we may operate on the assumption that the duration of the fixation is affected in some way using curly brackets. When we use curly brackets in new lines, it requires more visual attention to grasp the code. We can also observe that the overall average is highest for curly new lines and lowest for lines without curly. This suggests that using curly brackets in new lines requires more visual attention. On the other hand, all the algorithms that lack missing curly or without curly brackets them have the shortest fixation period on the AOI. However, for the time being, let's simply presume it to be the case; after doing statistical analysis, we will have absolute certainty about this matter.

4.3.2 Hypothesis testing visual attention:

This study requires statical analysis to assess constructed data for relevance. Research hypothesis testing will tell us the significance of results. In our study we have categorical data for our independent variable.

4.3.3 Inferential Statical analysis of visual attention (Fixation duration):

In this section we are going to find the answer for our last research question that is the Does the use of different style of curly brackets makes any difference in visual attention? To process the data for analysis we are going to divide data in three groups as we did for our past analysis. These groups are going to be Algo (WOC), Algo (WIC), Algo (CNL).

To continue the analysis, we will choose a confidence interval of 95% and an alpha value of 0.05. We will utilize one-way ANOVA because we have one independent variable three categories and with to do our analysis and one way ANOVA is the best fit in among other tests. First, we need to check the vital assumptions of ANOVA test which are normality of samples and equality of variance. We have checked normality of residual with Q-Q plot and Shipro Wilk test. Firstly, we will see explore Q-Q plot. With this we are going find the normality of residuals.

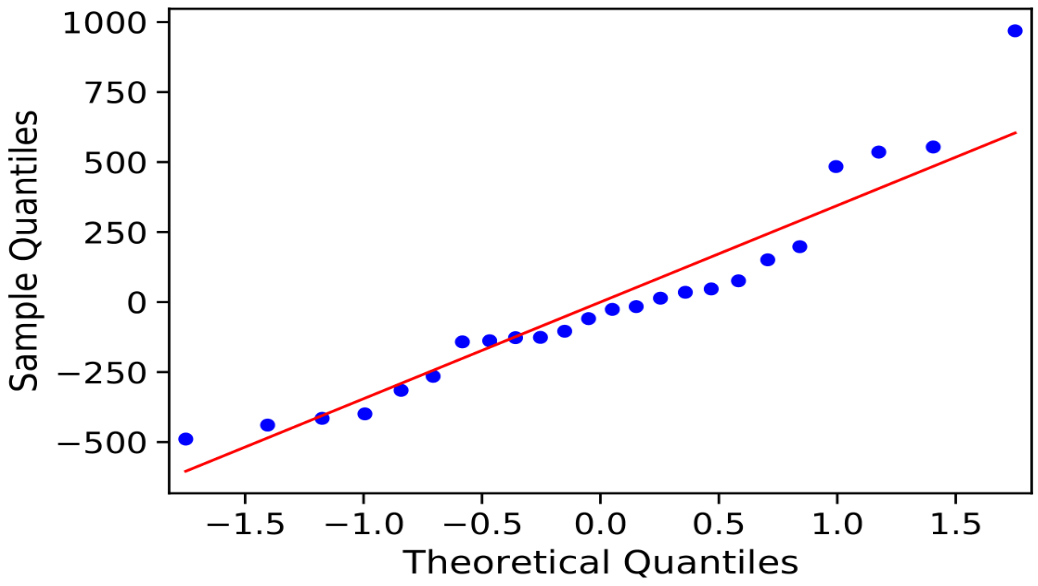


Figure 47. Q-Q plot to check the normality of residual for fixation

We can observe in Figure 47 that mostly data is normally distributed but some points are way above then the red line we can say that data is partially normally distributed. To check this more evidently, we have performed Shipro Wilk test where we found the Pvalue= 0.052 which is near to 0.05 but not significant enough and with Anderson Darling Test we found Pvalue=0.70 so we can say that it is not violating the ANOVA assumption for normality. Moreover, to check the equality of variance we have conducted LEVENE test of variance which gives us Pvalue of 0.090 which is far away then 0.05. Moreover, we have calculated the box plot to check the mean difference of the group

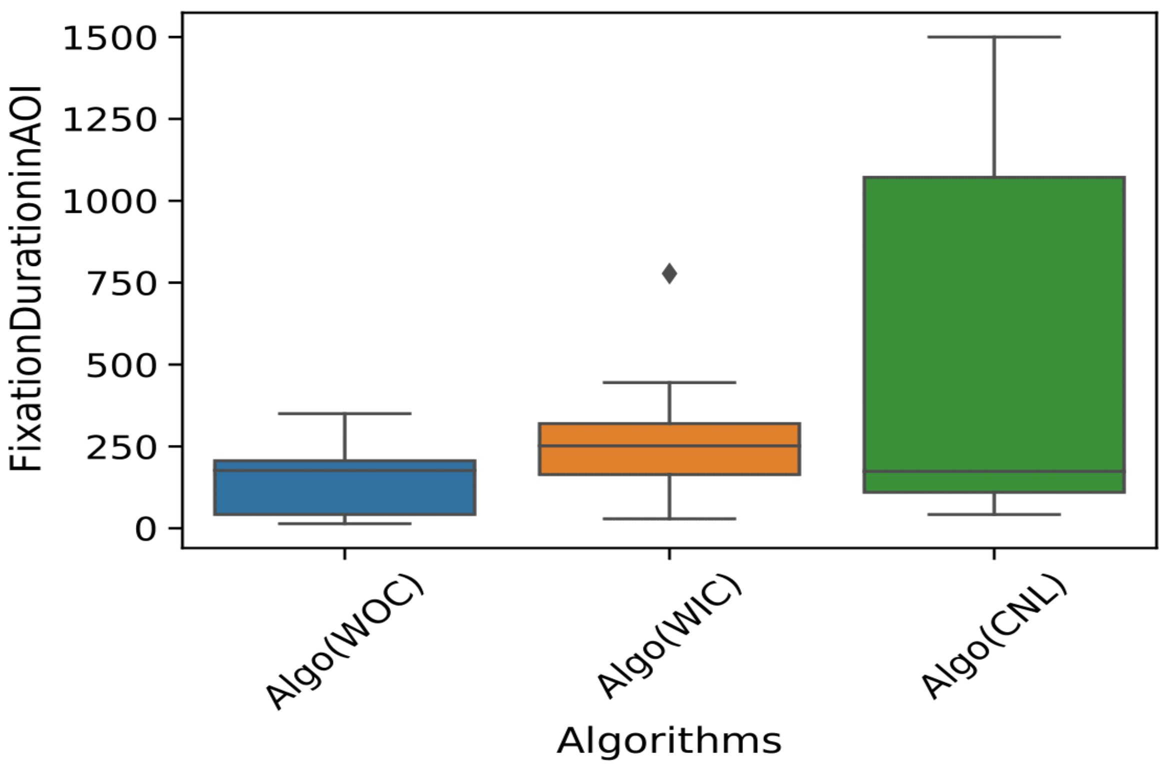


Figure 48. Boxplot comparing the response time mean for each group

Moreover, visualizing the data using a Boxplot reveals that the means of each group are not statistically distinguishable from one another and instead overlap. Boxplot means should not overlap, as we stated before, to get statistically significant findings.

Main effect	**Sum_sq**	**F value**	**PR(>F)**
Programming Style	5.87e+05	2.15	0.14
Residual	2.85e+06	-	-

Table 15. ANOVA test results

After testing the ANOVA test (Pvalue=0.14) and is quite strong evident that there is no significant difference in visual attention while using different style of curly brackets. Even though, in the findings of descriptive analysis we have found some results which stands out from another but that is not enough to be statical significant.

RQ2: Does the use of curly brackets in different style makes any difference in Visual attention ?
Answer: There's no statical significant evident that use of different style of curly brackets makes any different in the term of Visual attention.

5 Discussion

In response to the research question posted in section 1.2, we find that using curly brackets in different styles does not significantly affect the response time of participants during programme comprehension. Participants spent the most time understanding snippets where curly brackets began on the same line, known as the within curly brackets style, and the least time understanding snippets where curly brackets were removed from a single statement, known as the without curly brackets style. If we compare them individually, then BubbleSort(WIC) is an algorithm for which participants take the highest response time to comprehend, and ArraySum(WOC) is an algorithm where participants comprehend snippets in the quickest time.

Another finding we have in our study is that there is no significant effect on the accuracy of the participant while comprehending snippets with different styles of curly brackets. However, the overall accuracy percentage is highest when participants comprehend snippets where curly brackets begin on a new line, which means curly new line style snippets, and lowest when participants comprehend snippets with missing curly brackets, but there is not much difference between all three stylings for WOC styling (72%), WIC styling (74%), and CNL styling (75%). If we compare them individually, there are four algorithms which have a 100% accuracy rate: ArryaSum(WOC), ArryaSum(CNL) , NumberCheck (WIC), and BubbleSort(CNL). On the other hand, power (WOC) has the lowest accuracy percentage.

For the second research question in section 1.2, we have calculated the average fixation duration in the area of interest, which is curly brackets, in our study and compared them for different categories. We find that there is no significant effect on the participant's visual attention when using different styles of curly brackets in programme comprehension. But we have seen some interesting observations. We have found the algorithms that have more lines in code and curly brackets starting from a new line have the highest average fixation duration. For example, MultiplyMatrix(CNL)(1500ms),ContainsSubstring(CNL)(1085ms),and BinaryConversion(CNL). (1067ms) have the highest average response time on AOI . On the other hand, the lowest average fixation duration in AOI is for ArraySum(WOC). If we talk about overall fixation duration, then algorithms where curly brackets start from a new line have the highest average fixation duration in AOI and algorithms where curly brackets are missing have the lowest average fixation duration in AOI.

5.1 Threats to Validity

- Internal validity:

Internal validity refers to the presence of other factors besides the main factor that might have an effect on the results. There are some internal threats to validity in over study, such as in our study, we used a table mounted eye tracker but the participant's head was not stabilized, which may have impacted visual effect dependent verifiability. Moreover, the length of the code was limited in our study, and we have seen some different values for lengthy codes in our discussion. This also might have an effect on internal validity.

- External validity:

External validity is about how our results can be used in the real world. In our study, there are some threats to the external validity. First, we selected participants just based on their academic background, which is software engineering, but we did not compare them regarding their experience, so our study cannot be generalized in terms of the experience of the programmer. Moreover, in our study, we used a limit of the number of lines of code, which was a maximum of 35 lines. We cannot generalize our study results for all lengths of code.

6 Conclusion and Future work

An eye tracking study to investigate the effects of curly brackets in different style on response time, accuracy, and visual attention in program comprehension. To analyze visual attention, we used average response time on Area of interest (AOI). The task was to comprehend some snippets and choose the answer for that snippet. No difference was found between use of different style of curly brackets with respect to response time, accuracy, and visual attention. Although we have found some interesting results while analyzing visual attention as more lengthy code with curly new line has highest average fixation duration in AOI, but it was not enough to be statically significant.

Our study was the first eye tracking study to investigate the effect of curly brackets on programme comprehension, and we did not find any significant difference in our study. There are so many things we can do with study in the future. First, we can conduct the eye tracking experiment with a stronger setup, like providing head stabilization, studying with more participants, and including experience as the independent variable. Moreover, we can use more lengthy and complex snippets, so the study will have strong external validity. EEG and fMRI techniques can be used to get more ideas about the congenital processes of the programmer.

References

[1] Siegmund, Janet and Jana Schumann. “Confounding parameters on program comprehension: a literature survey”. In: Empirical Software Engineering 20.4 (2015), pp. 1159–1192. issn: 1573-7616.

[2] Peitek, Norman, et al., ed. Simultaneous measurement of program comprehension with fmri and eye tracking: A case study. 2018.

[3] Peitek N, Siegmund J, Parnin C, Apel S, Brechmann A. Toward conjoint analysis of simultaneous eye-tracking and fMRI data for program-comprehension studies. InProceedings of the Workshop on Eye Movements in Programming 2018 Jun 15 (pp. 1-5).

[4] Letovsky S. Cognitive processes in program comprehension. Journal of Systems and software. 1987 Dec 1;7(4):325-39.

[5] Siegmund, Janet, ed. Program comprehension: Past, present, and future. IEEE, 2016. isbn: 1509018557.

[6] Sharif B, Maletic JI. An eye tracking study on camelcase and under_score identifier styles. In2010 IEEE 18th International Conference on Program Comprehension 2010 Jun 30 (pp. 196-205). IEEE.

[7] Peitek N, Apel S, Parnin C, Brechmann A, Siegmund J. Program comprehension and code complexity metrics: An fmri study. In2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) 2021 May 22 (pp. 524-536). IEEE.

[8] Arooba A, Peitek N, Apel S, Mucke J, Siegmund J. Understanding Comprehension of Iterative and Recursive Programs with Remote Eye Tracking.

[9] Norman Peitek, Janet Siegmund, and Sven Apel. 2020. What Drives the Reading Order of Programmers? An Eye Tracking Study. In Proceedings of 28th International Conference on Program Comprehension, Seoul, Republic of Korea, October 5–6, 2020 (ICPC ’20), 12 pages. https://doi.org/10.1145/3387904.3389279

[10]Gopstein D, Iannacone J, Yan Y, DeLong L, Zhuang Y, Yeh MK, Cappos J. Understanding misunderstandings in source code. InProceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering 2017 Aug 21 (pp. 129-139).

[11] Incident report on memory leak caused by Cloudflare parser bug (2017) https://blog.cloudflare.com/incident-report-on-memory-leak-caused-by-cloudflare-parser-bug/

[12] https://stackoverflow.com/questions/2125066/is-it-a-bad-practice-to-use-an-if statement-without-curly-braces

[13] Bansal AK. Introduction to programming languages. CRC Press; 2014.

[14] Herbert, Schildt. "C: The Complete Reference." (2000).

[15] Schildt, Herbert. "The complete reference Java." (2020).

[16] Ritchie DM. The development of the C language. ACM Sigplan Notices. 1993 Apr 20;28(3):201-8.

[17] Matúš. Sulír, ed. Program comprehension: A short literature review. 2015

[18] Wilhelm Max Wundt. Grundzüge der physiologischen Psychologie. W. Engelman, 1874.

[19] Amela Karahasanović et al. “Comparing of feedback-collection and think-aloud methods in program comprehension studies”. In: Behaviour & Information Technology 28.2 (2009), pp. 139–164. issn: 0144-929X.

[20] Ben Shneiderman. “Exploratory experiments in programmer behavior”. In:International Journal of Computer & Information Sciences 5.2 (1976), pp. 123–143. issn: 1573-7640.

[21] Nancy Pennington. “Stimulus structures and mental representations in expert comprehension of computer programs”. In: Cognitive psychology 19.3 (1987),pp. 295–341. issn: 0010-0285. Alastair Dunsmore and Marc Roper. “A comparative evaluation of program comprehension measures”. In: The Journal of Systems and Software 52.3 (2000), pp. 121–129.

[22] Keith Rayner. “Eye movements in reading and information processing”. In: Psychological bulletin 85.3 (1978), p. 618. issn: 1939-1455.

[23] Punde PA, Jadhav ME, Manza RR. A study of eye tracking technology and its applications. In2017 1st International Conference on Intelligent Systems and Information Management (ICISIM) 2017 Oct 5 (pp. 86-90). IEEE.

[24] Kangas J, Rantala J, Majaranta P, Isokoski P, Raisamo R. Haptic feedback to gaze events. InProceedings of the Symposium on Eye Tracking Research and Applications 2014 Mar 26 (pp. 11-18).

[25] Hutton SB. Eye tracking methodology. In Eye Movement Research 2019 (pp. 277-308). Springer, Cham.

[26] Grossman RB, Zane E, Mertens J, Mitchell T. Facetime vs. Screentime: Gaze patterns to live and video social stimuli in adolescents with ASD. Scientific reports. 2019 Sep 2;9(1):1-0.

[27] https://www.ergoneers.com/en/en/hardware/eye-tracking/head-mounted/

[28] Blascheck T, Kurzhals K, Raschke M, Burch M, Weiskopf D, Ertl T. Visualization of eye tracking data: A taxonomy and survey. InComputer Graphics Forum 2017 Dec (Vol. 36, No. 8, pp. 260-284).

[29] Sharafi Z, Shaffer T, Sharif B, Guéhéneuc YG. Eye-tracking metrics in software engineering. In2015 Asia-Pacific Software Engineering Conference (APSEC) 2015 Dec 1 (pp. 96-103). IEEE.

[30] Punde PA, Jadhav ME, Manza RR. A study of eye tracking technology and its applications. In2017 1st International Conference on Intelligent Systems and Information Management (ICISIM) 2017 Oct 5 (pp. 86-90). IEEE.

[31] Salvucci DD, Goldberg JH. Identifying fixations and saccades in eye-tracking protocols. InProceedings of the 2000 symposium on Eye tracking research & applications 2000 Nov 8 (pp. 71-78).

[32] Ghose U, Srinivasan AA, Boyce WP, Xu H, Chng ES. PyTrack: An end-to-end analysis toolkit for eye tracking. Behavior research methods. 2020 Dec;52(6):2588-603.

[33] Sharafi Z, Soh Z, Guéhéneuc YG. A systematic literature review on the usage of eye-tracking in software engineering. Information and Software Technology. 2015 Nov 1;67:79-107.

[34] Crosby ME, Stelovsky J. How do we read algorithms? A case study. Computer. 1990 Jan;23(1):25-35.

[35] Bednarik R, Tukiainen M. An eye-tracking methodology for characterizing program comprehension processes. InProceedings of the 2006 symposium on Eye tracking research & applications 2006 Mar 27 (pp. 125-132).

[36] Busjahn T, Bednarik R, Begel A, Crosby M, Paterson JH, Schulte C, Sharif B, Tamm S. Eye movements in code reading: Relaxing the linear order. In2015 IEEE 23rd International Conference on Program Comprehension 2015 May 18 (pp. 255-265). IEEE.

[37] Binkley D, Davis M, Lawrie D, Maletic JI, Morrell C, Sharif B. The impact of identifier style on effort and comprehension. Empirical software engineering. 2013 Apr;18(2):219-76.

[38] Duru HA, Çakır MP, İşler V. How does software visualization contribute to software comprehension? A grounded theory approach. International Journal of Human-Computer Interaction. 2013 Nov 2;29(11):743-63.

[39] Fritz T, Begel A, Müller SC, Yigit-Elliott S, Züger M. Using psycho-physiological measures to assess task difficulty in software development. InProceedings of the 36th international conference on software engineering 2014 May 31 (pp. 402-413).

[40] Rodeghero P, McMillan C, McBurney PW, Bosch N, D'Mello S. Improving automated source code summarization via an eye-tracking study of programmers. InProceedings of the 36th international conference on Software engineering 2014 May 31 (pp. 390-401).

[41]Ishida T, Uwano H. Synchronized analysis of eye movement and EEG during program comprehension. In2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP) 2019 May 27 (pp. 26-32). IEEE.

[42]Peitek N, Siegmund J, Apel S, Kästner C, Parnin C, Bethmann A, Leich T, Saake G, Brechmann A. A look into programmers' heads. IEEE Transactions on Software Engineering. 2018 Aug 6;46(4):442-62.

[43] Peirce J, Gray JR, Simpson S, MacAskill M, Höchenberger R, Sogo H, Kastman E, Lindeløv JK. PsychoPy2: Experiments in behavior made easy. Behavior research methods. 2019 Feb;51(1):195-203.

[44] Charness G, Gneezy U, Kuhn MA. Experimental methods: Between-subject and within-subject design. Journal of economic behavior & organization. 2012 Jan 1;81(1):1-8.

[45] Chivers ID, Sleightholme J. Introduction to programming with Fortran. Berlin: Springer; 2018 Aug 21.

[46] Wirth N, Hoare CA. A contribution to the development of ALGOL. Communications of the ACM. 1966 Jun 1;9(6):413-32.

[47] Woodger M. An introduction to ALGOL 60. The Computer Journal. 1960 Jan 1;3(2):67-75.

[48] Richards M. BCPL: A tool for compiler writing and system programming. InProceedings of the May 14-16, 1969, spring joint computer conference 1969 May 14 (pp. 557-566).

[49] Johnson SC, Kernighan BW. The programming language B. Murray Hill, New Jersey: Bell Laboratories; 1973 Jan.

[50] Hanford KV, Jones CB. Dynamic syntax: A concept for the definition of the syntax of programming languages. Annual review in automatic programming. 1973 Jan 1;7:115-42.

[51] Slonneger K, Kurtz BL. Formal syntax and semantics of programming languages. Reading: Addison-Wesley; 1995 Jan.

[52] Floyd RW. The syntax of programming languages-a survey. IEEE Transactions on Electronic Computers. 1964 Aug(4):346-53.

[53] TURBAK F, GIFFORD D. Design Concepts in Programming Languages.

[54]Bansal AK. Introduction to programming languages. CRC Press; 2014.

[55] Agrawal H, DeMillo RA, Hathaway R, Hsu W, Hsu W, Krauser EW, Martin RJ, Mathur AP, Spafford E. Design of mutant operators for the C programming language. Technical Report SERC-TR-41-P, Software Engineering Research Center, Purdue University; 1989 Mar 20.

[56] Strachey C. Fundamental concepts in programming languages. Higher-order and symbolic computation. 2000 Apr;13(1):11-49.

[57] McMaster K, Sambasivam S, Wolthuis S. Teaching programming style with ugly code. InProceedings of the Information Systems Educators Conference ISSN 2013 (Vol. 2167, p. 1435).

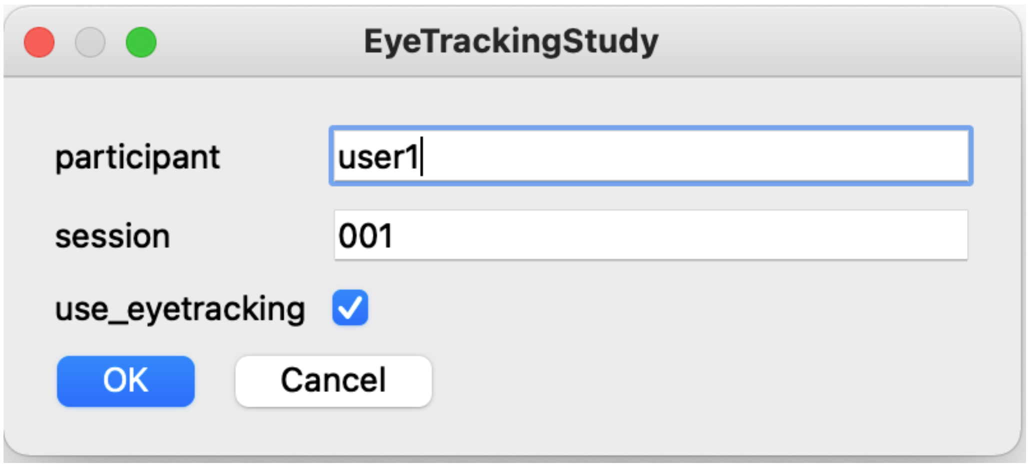
[58] https://stitcher.io/blog/where-a-curly-bracket-belongs

[59] **https://www.kernel.org/doc/html/v4.10/process/coding-style.html**

[60] https://www.originate.com/thinking/a-unique-guide-to-syntax-from-a-developers-v

QUESTIONNAIRE

This section provides complete sample questionnaire used to get a information from the participant. We have provided a sample test for combination 3.

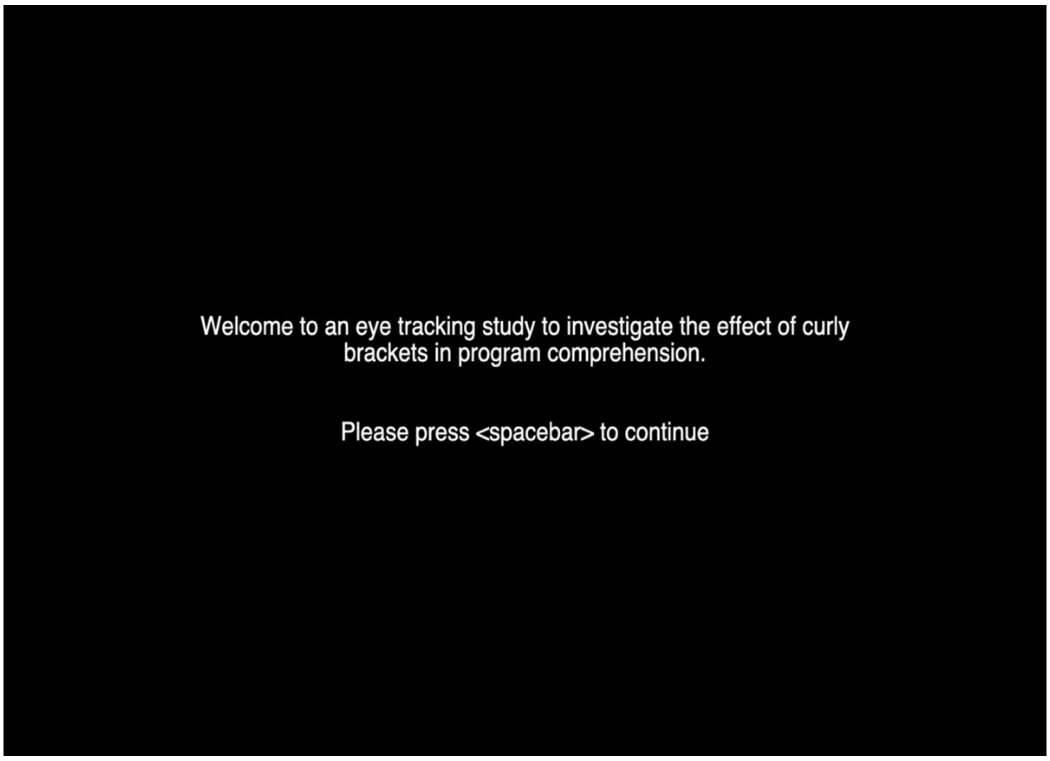

EyeTrackingStudy

participant: user1

session: 001

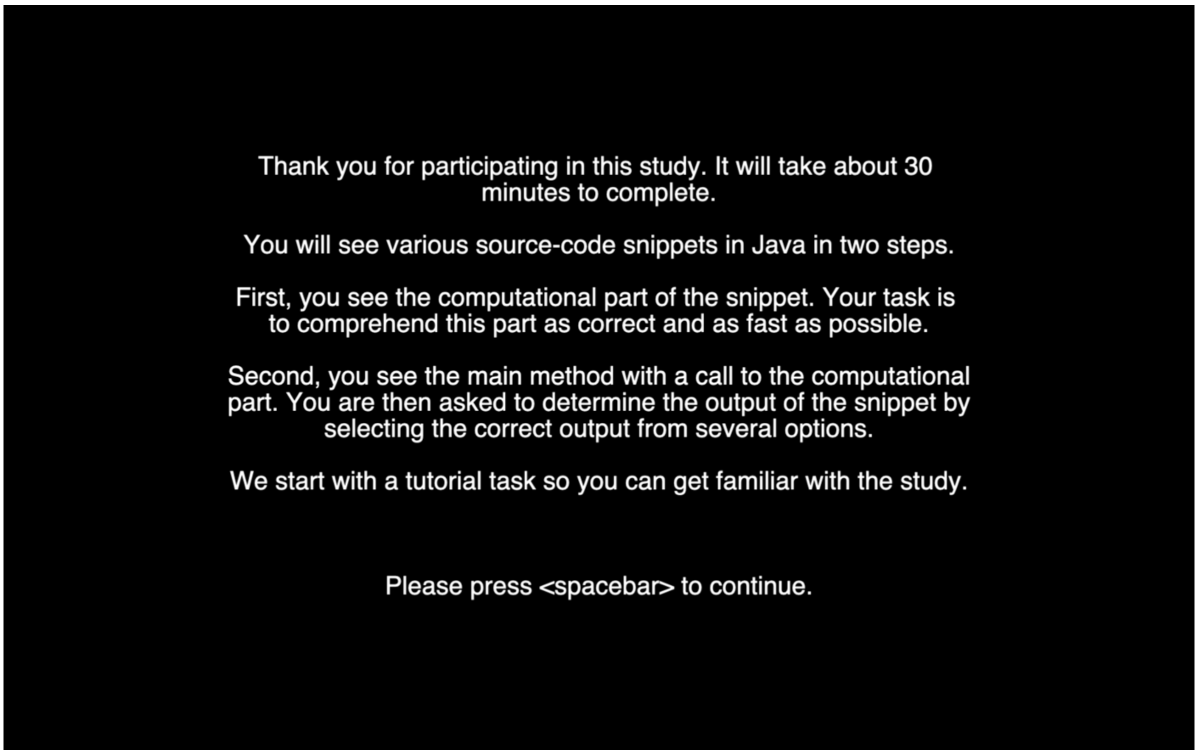
use_eyetracking: ✓

Figure 49. First step user insert their information an click on user_eye tracking


Welcome to an eye tracking study to investigate the effect of curly brackets in program comprehension.

Please press <spacebar> to continue

Figure 50. Welcome Screen for participant


Thank you for participating in this study. It will take about 30 minutes to complete.

You will see various source-code snippets in Java in two steps.

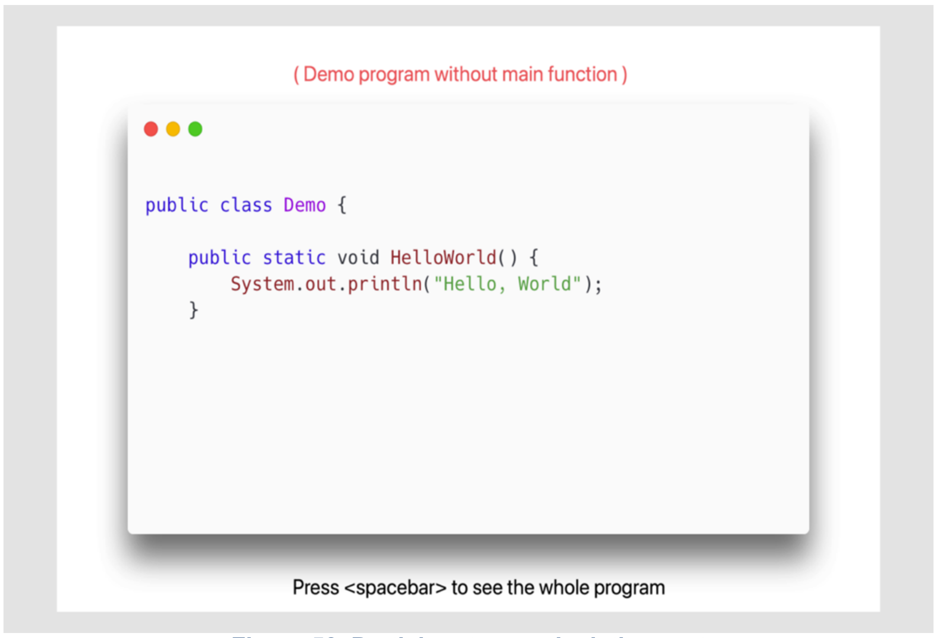
First, you see the computational part of the snippet. Your task is to comprehend this part as correct and as fast as possible.

Second, you see the main method with a call to the computational part. You are then asked to determine the output of the snippet by selecting the correct output from several options.

We start with a tutorial task so you can get familiar with the study.

Please press <spacebar> to continue.

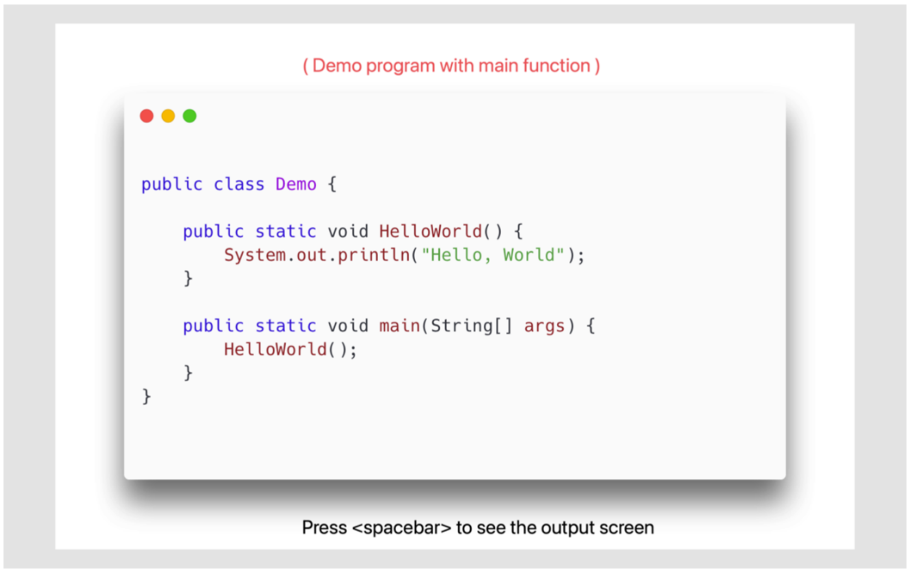
Figure 51. Instructions for the participant.


(Demo program without main function)

```
public class Demo {

    public static void HelloWorld() {
        System.out.println("Hello, World");
    }
```

Press <spacebar> to see the whole program

Figure 52. Participant see calculation part


(Demo program with main function)

```
public class Demo {

    public static void HelloWorld() {
        System.out.println("Hello, World");
    }

    public static void main(String[] args) {
        HelloWorld();
    }
}
```

Press <spacebar> to see the output screen

Figure 53. Participant see after whole program after pressing spacebar

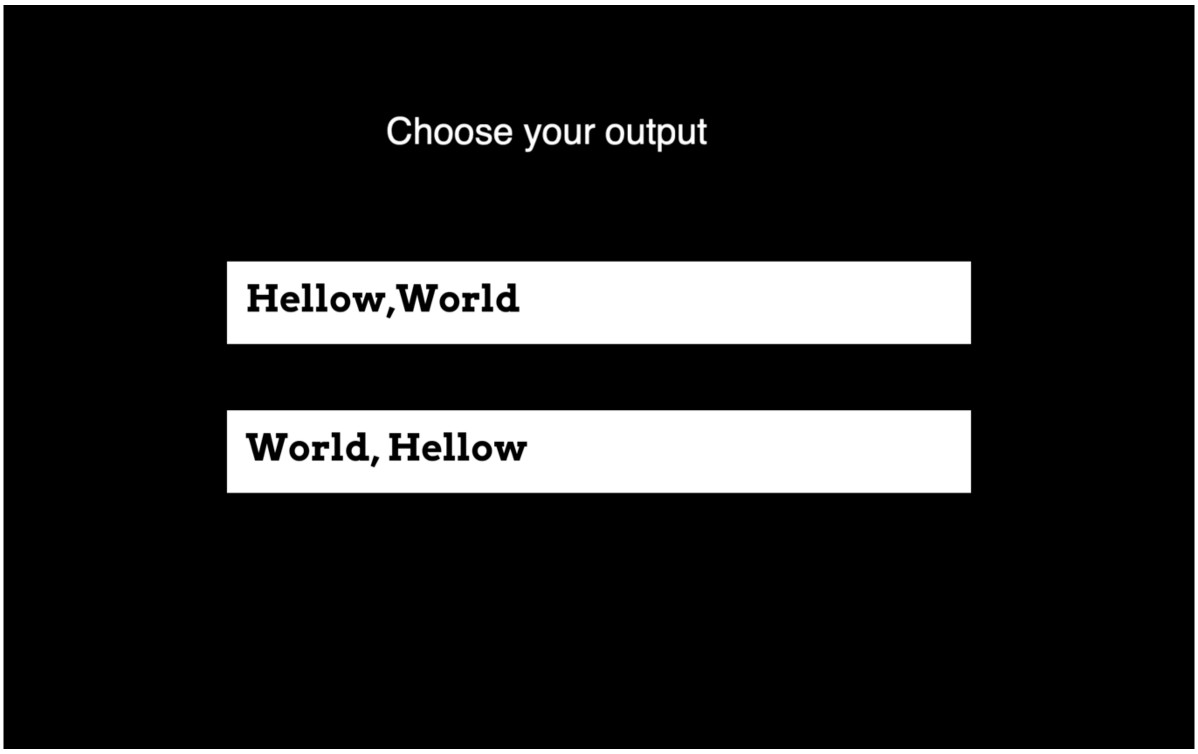
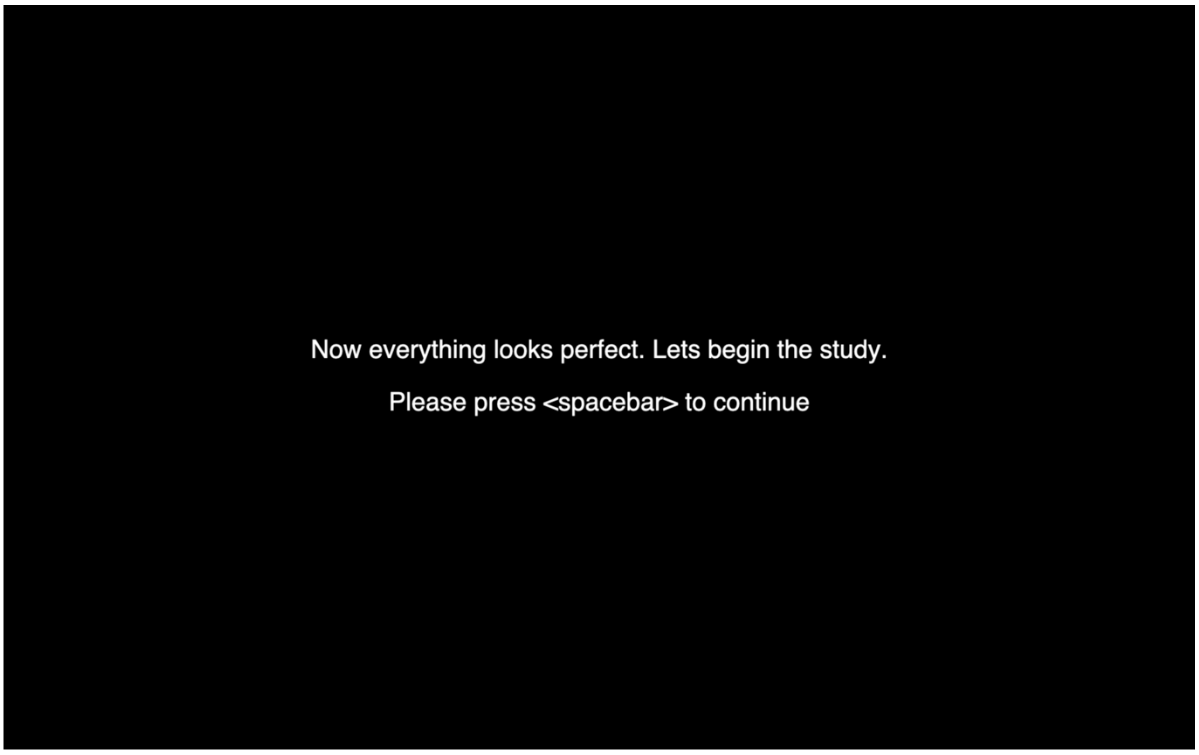
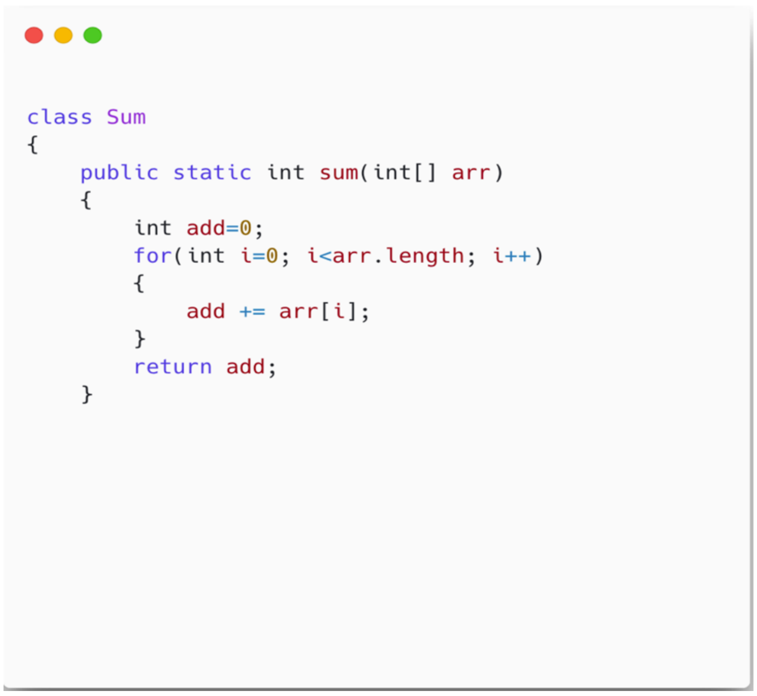


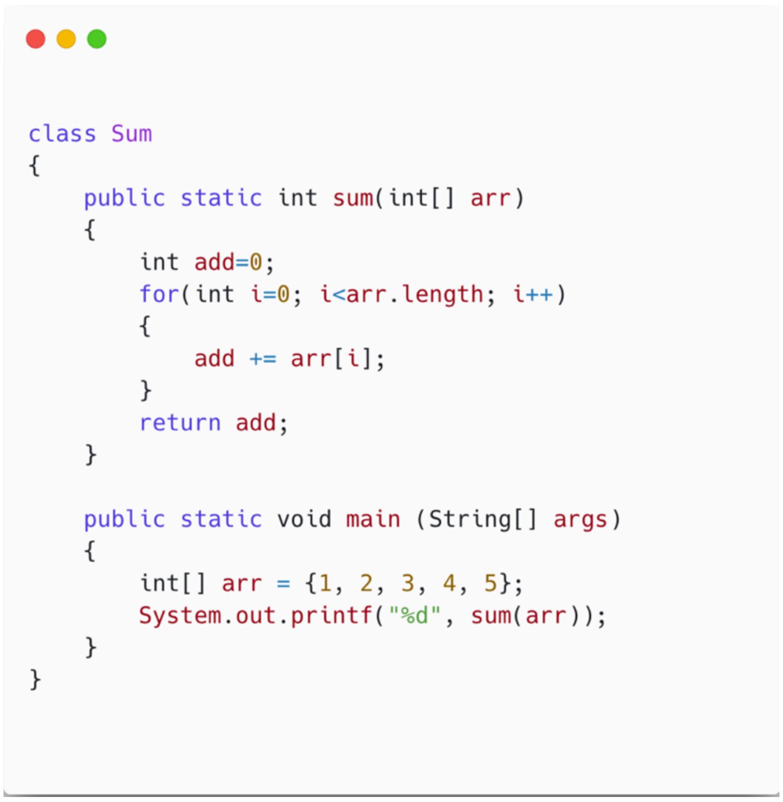
Figure 54. **Output screen for the participant for demo program**


Now everything looks perfect. Lets begin the study.

Please press <spacebar> to continue

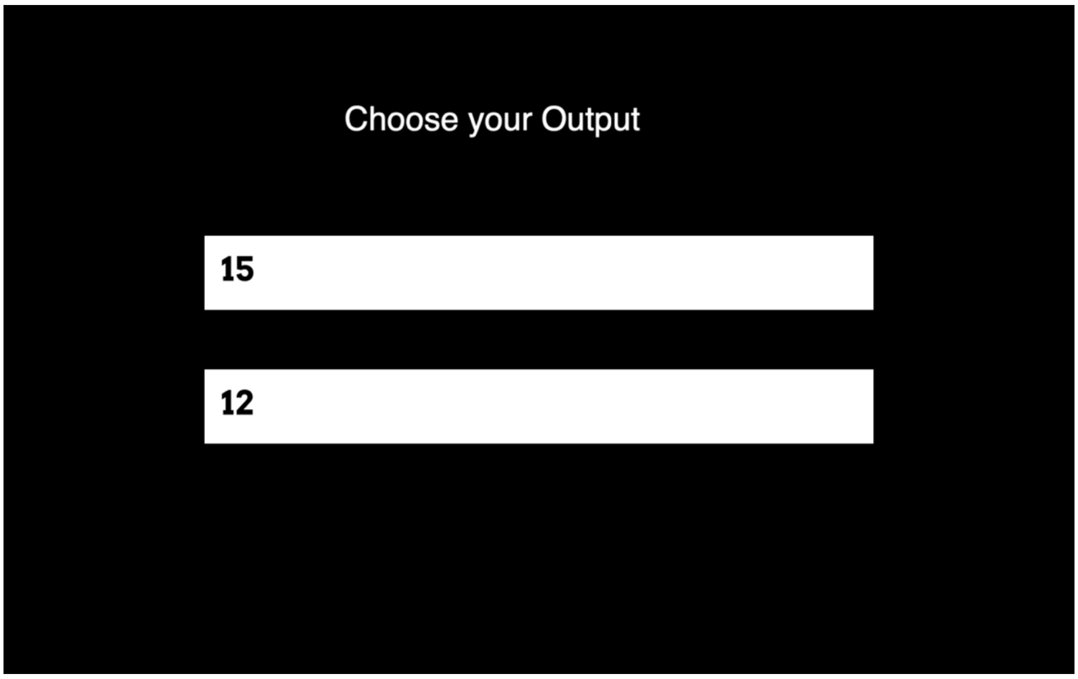


```java
class Sum
{
    public static int sum(int[] arr)
    {
        int add=0;
        for(int i=0; i<arr.length; i++)
        {
            add += arr[i];
        }
        return add;
    }
```

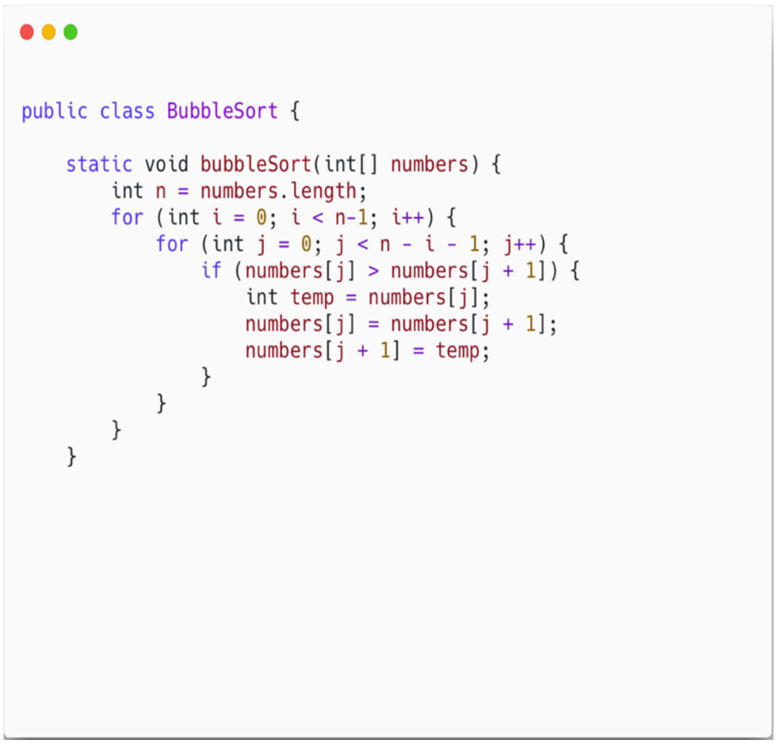


```java
class Sum
{
    public static int sum(int[] arr)
    {
        int add=0;
        for(int i=0; i<arr.length; i++)
        {
            add += arr[i];
        }
        return add;
    }

    public static void main (String[] args)
    {
        int[] arr = {1, 2, 3, 4, 5};
        System.out.printf("%d", sum(arr));
    }
}
```

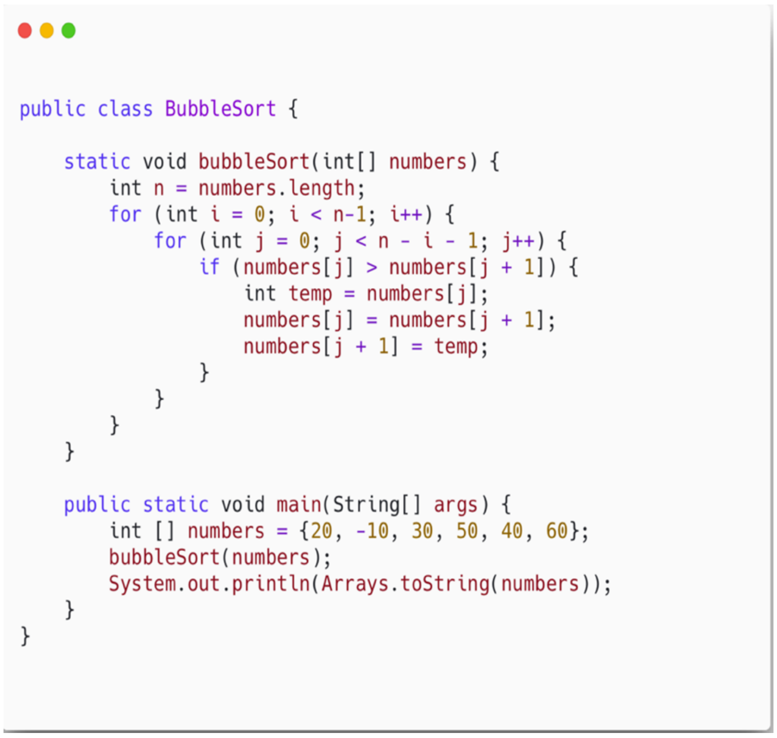
Program 1. Array Sum



Output screen for ArraySum

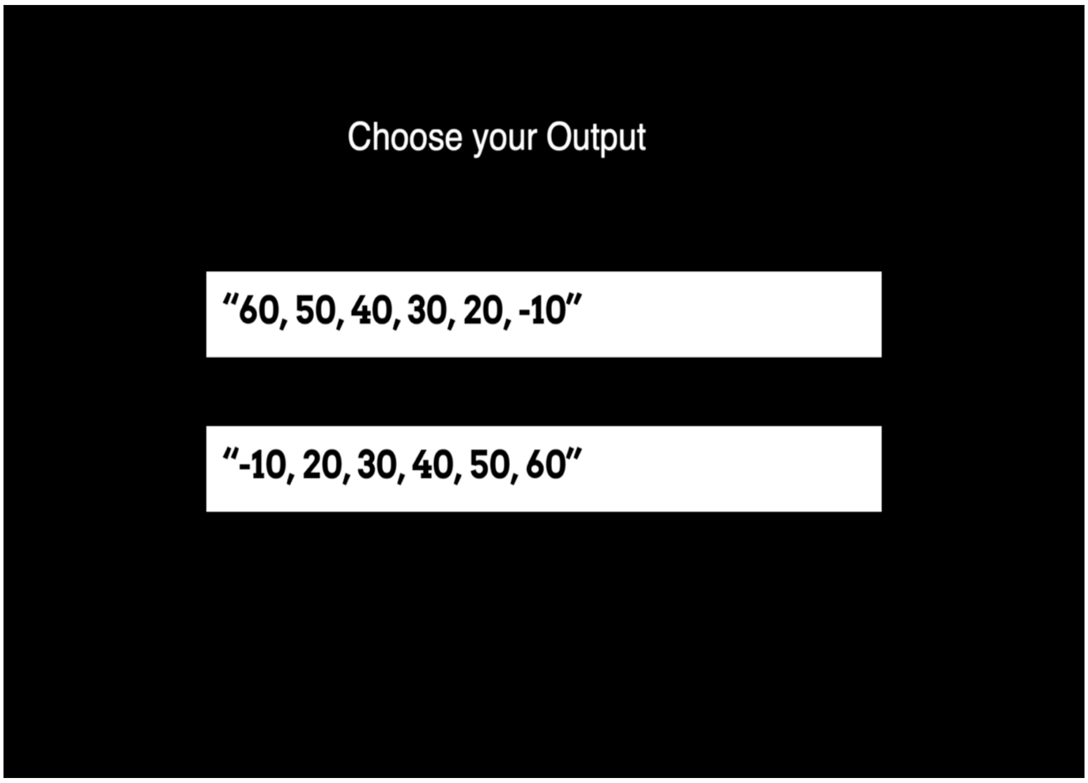


```
public class BubbleSort {

    static void bubbleSort(int[] numbers) {
        int n = numbers.length;
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (numbers[j] > numbers[j + 1]) {
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }
    }
```



```
public class BubbleSort {

    static void bubbleSort(int[] numbers) {
        int n = numbers.length;
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (numbers[j] > numbers[j + 1]) {
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        int [] numbers = {20, -10, 30, 50, 40, 60};
        bubbleSort(numbers);
        System.out.println(Arrays.toString(numbers));
    }
}
```

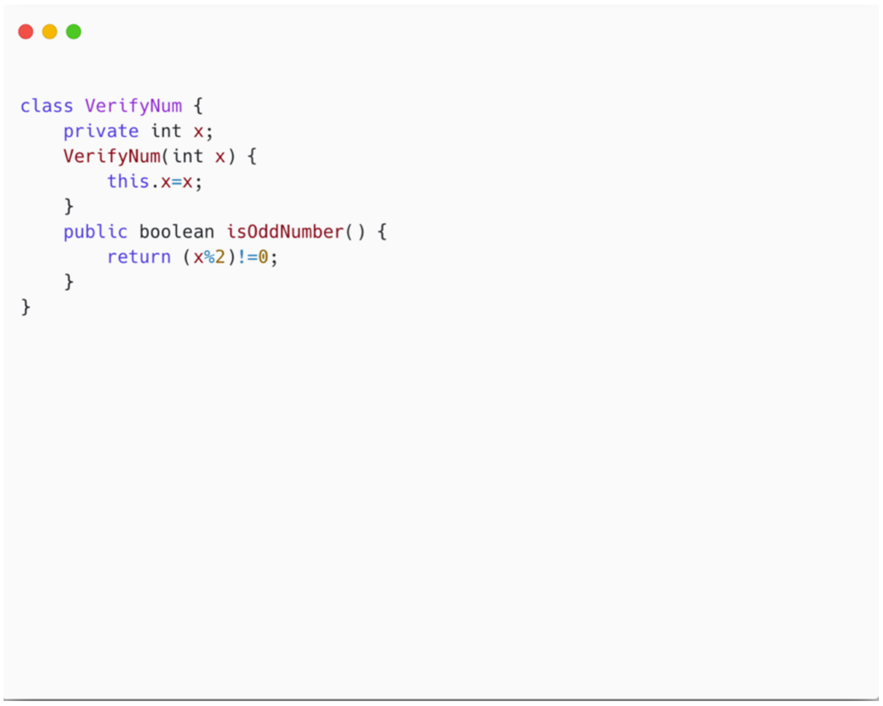
Program 2. BubbleSort


Choose your Output

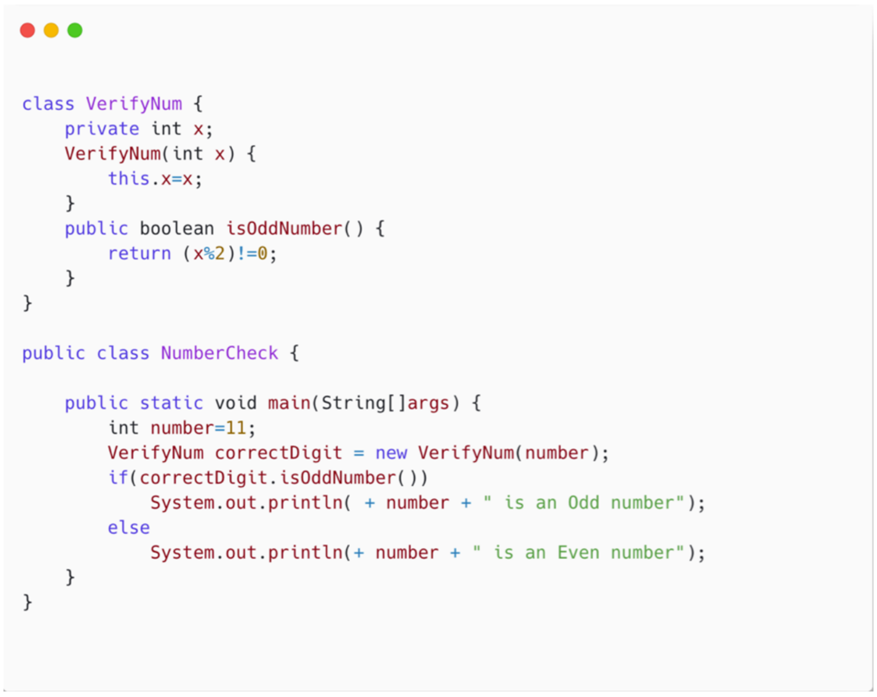
“60, 50, 40, 30, 20, -10”

“-10, 20, 30, 40, 50, 60”

Output screen for BubbleSort

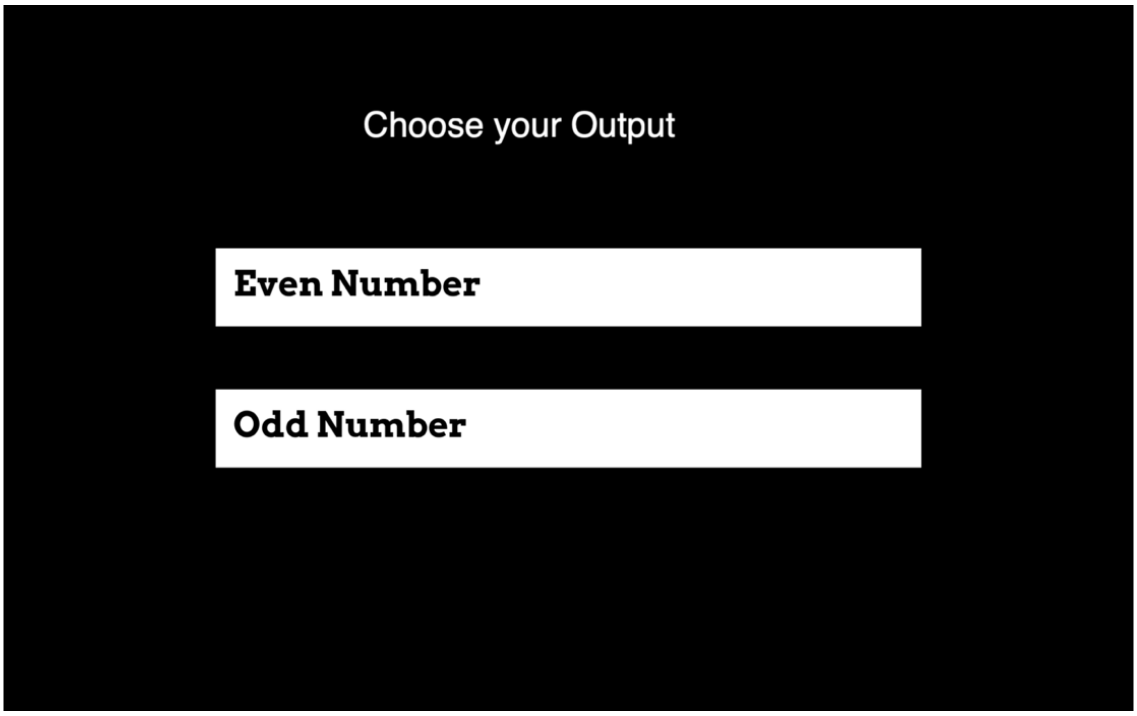


```java
class VerifyNum {
    private int x;
    VerifyNum(int x) {
        this.x=x;
    }
    public boolean isOddNumber() {
        return (x%2)!=0;
    }
}
```

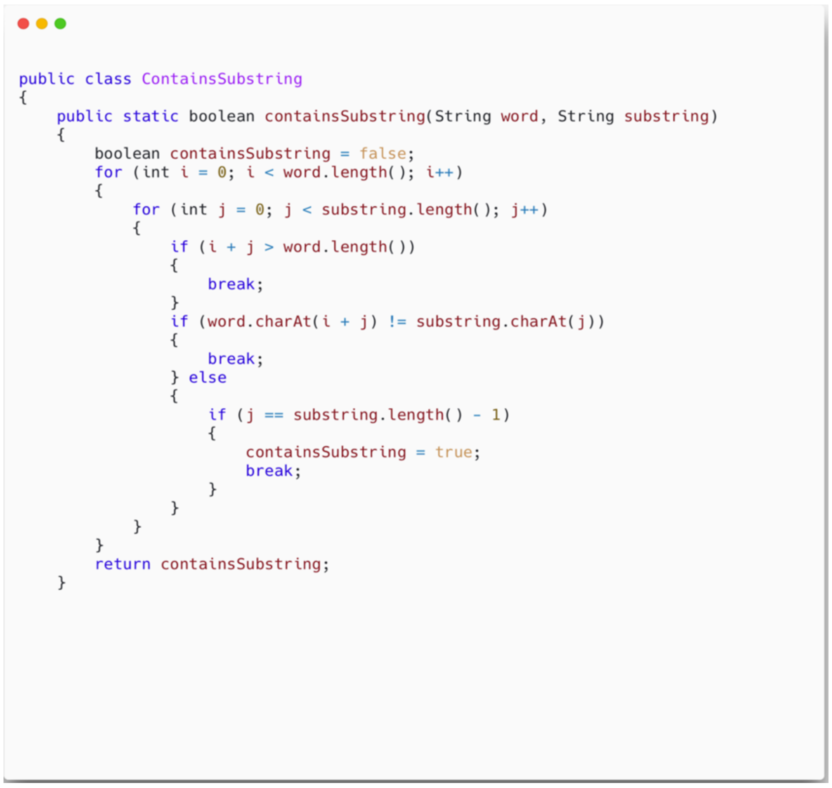


```java
class VerifyNum {
    private int x;
    VerifyNum(int x) {
        this.x=x;
    }
    public boolean isOddNumber() {
        return (x%2)!=0;
    }
}

public class NumberCheck {

    public static void main(String[]args) {
        int number=11;
        VerifyNum correctDigit = new VerifyNum(number);
        if(correctDigit.isOddNumber())
            System.out.println( + number + " is an Odd number");
        else
            System.out.println(+ number + " is an Even number");
    }
}
```

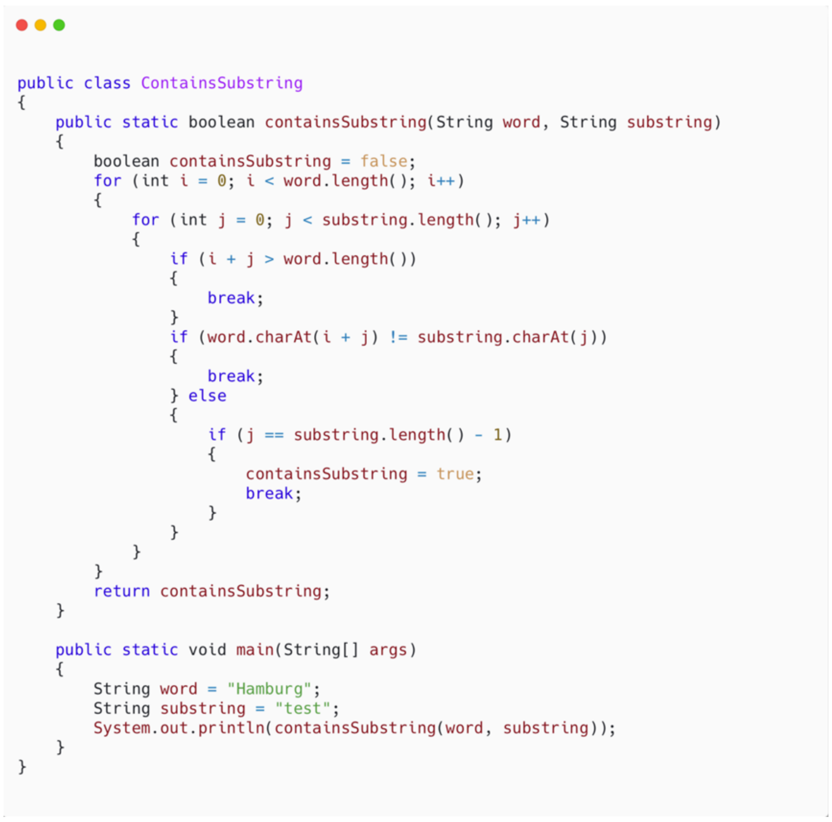
Program 3. NumberCheck



Output screen for NumberCheck

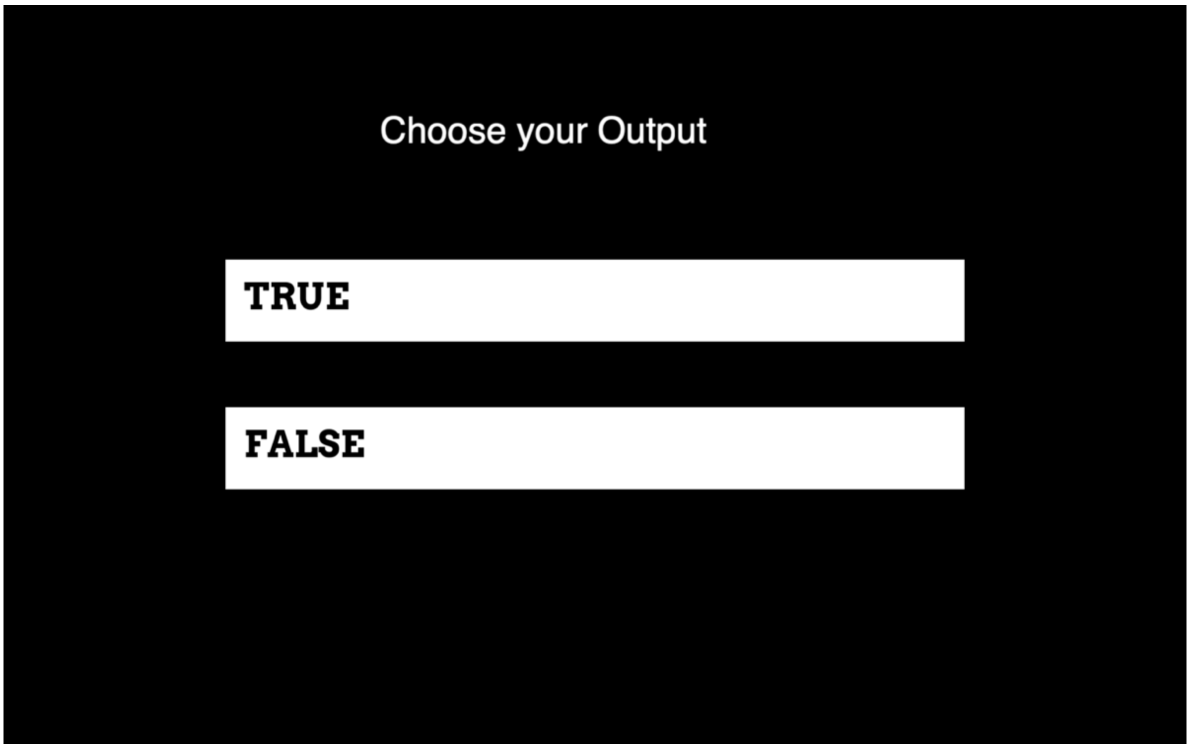


```
public class ContainsSubstring
{
    public static boolean containsSubstring(String word, String substring)
    {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++)
        {
            for (int j = 0; j < substring.length(); j++)
            {
                if (i + j > word.length())
                {
                    break;
                }
                if (word.charAt(i + j) != substring.charAt(j))
                {
                    break;
                } else
                {
                    if (j == substring.length() - 1)
                    {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }
```

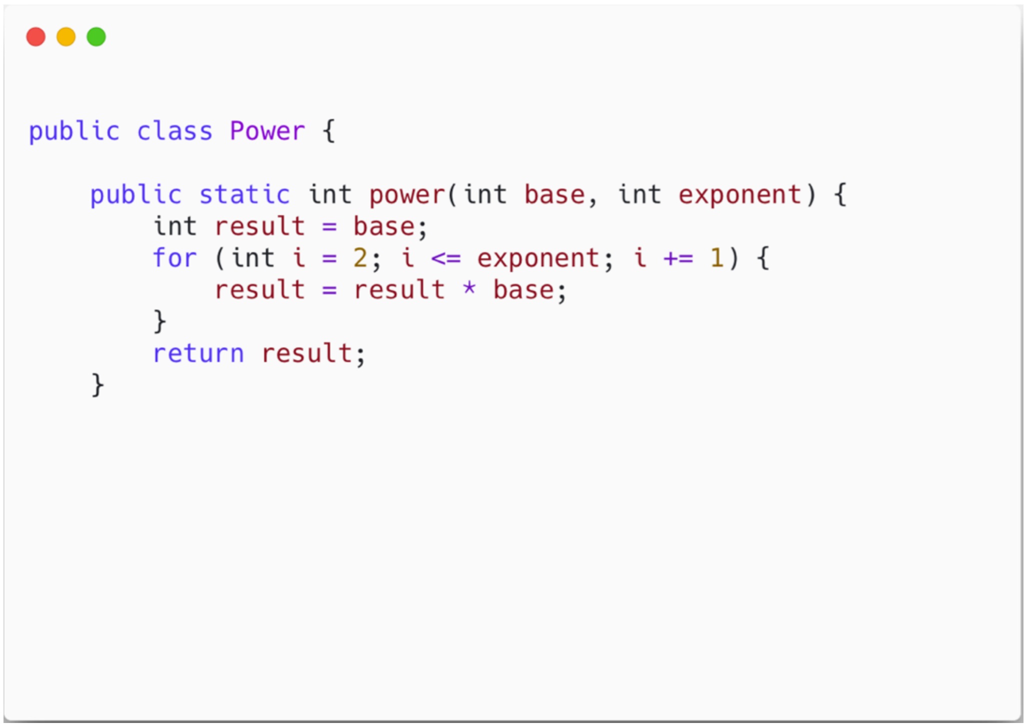


```
public class ContainsSubstring
{
    public static boolean containsSubstring(String word, String substring)
    {
        boolean containsSubstring = false;
        for (int i = 0; i < word.length(); i++)
        {
            for (int j = 0; j < substring.length(); j++)
            {
                if (i + j > word.length())
                {
                    break;
                }
                if (word.charAt(i + j) != substring.charAt(j))
                {
                    break;
                } else
                {
                    if (j == substring.length() - 1)
                    {
                        containsSubstring = true;
                        break;
                    }
                }
            }
        }
        return containsSubstring;
    }

    public static void main(String[] args)
    {
        String word = "Hamburg";
        String substring = "test";
        System.out.println(containsSubstring(word, substring));
    }
}
```

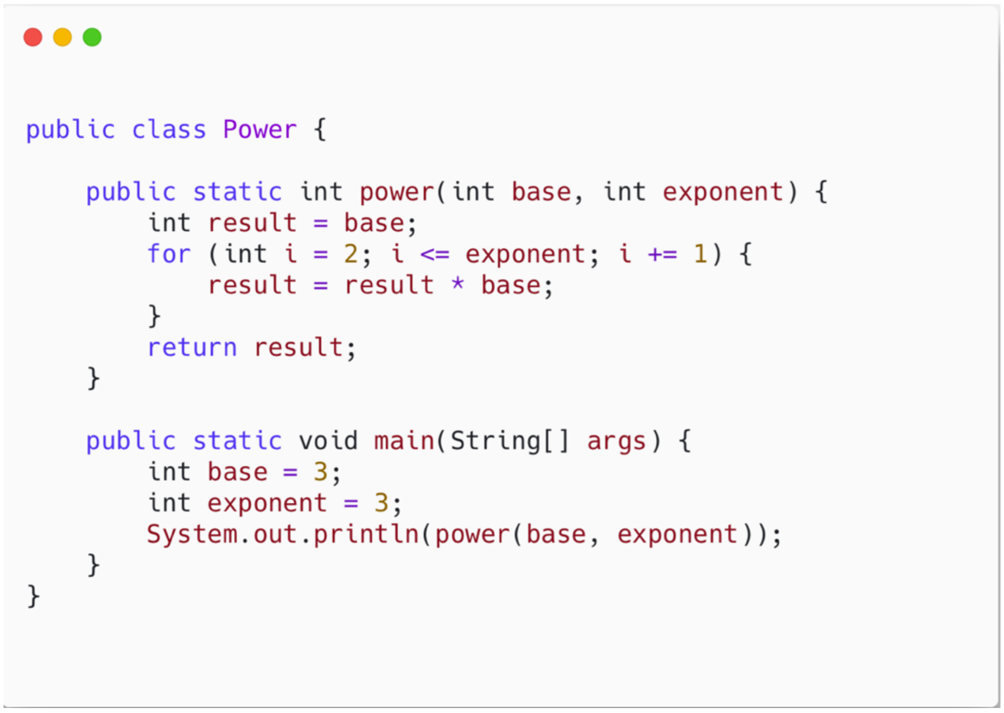
Program 4. ContainsSubstring



Output screen for ContainsSubstring

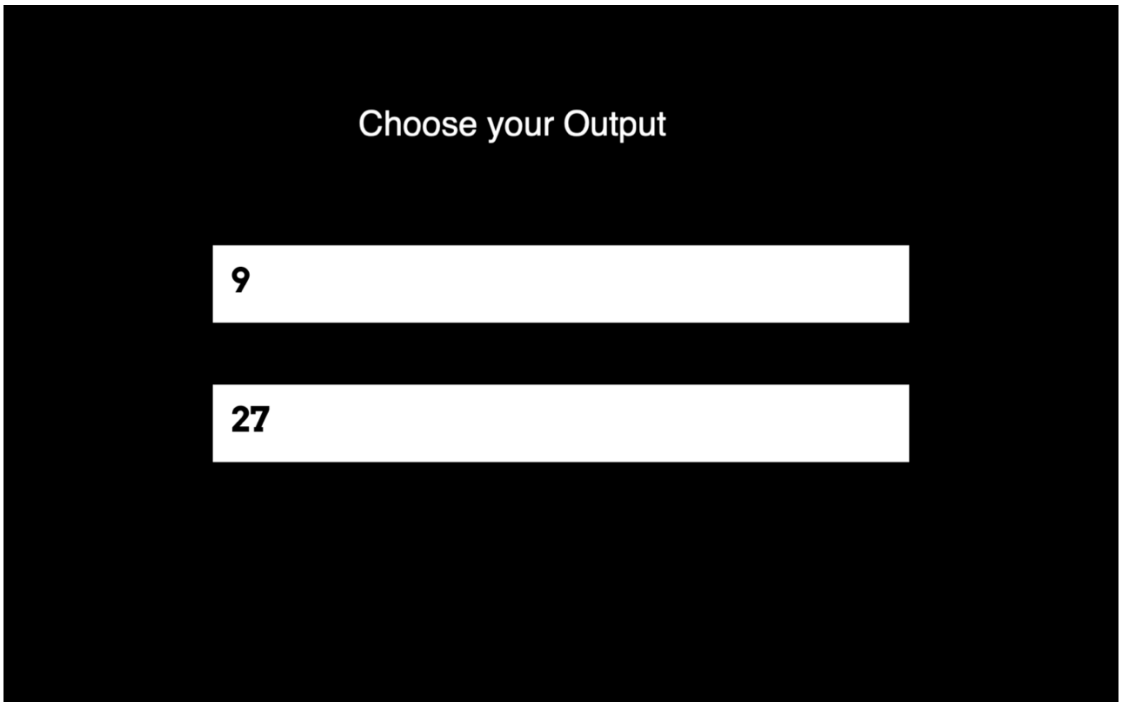


```
public class Power {

    public static int power(int base, int exponent) {
        int result = base;
        for (int i = 2; i <= exponent; i += 1) {
            result = result * base;
        }
        return result;
    }
```

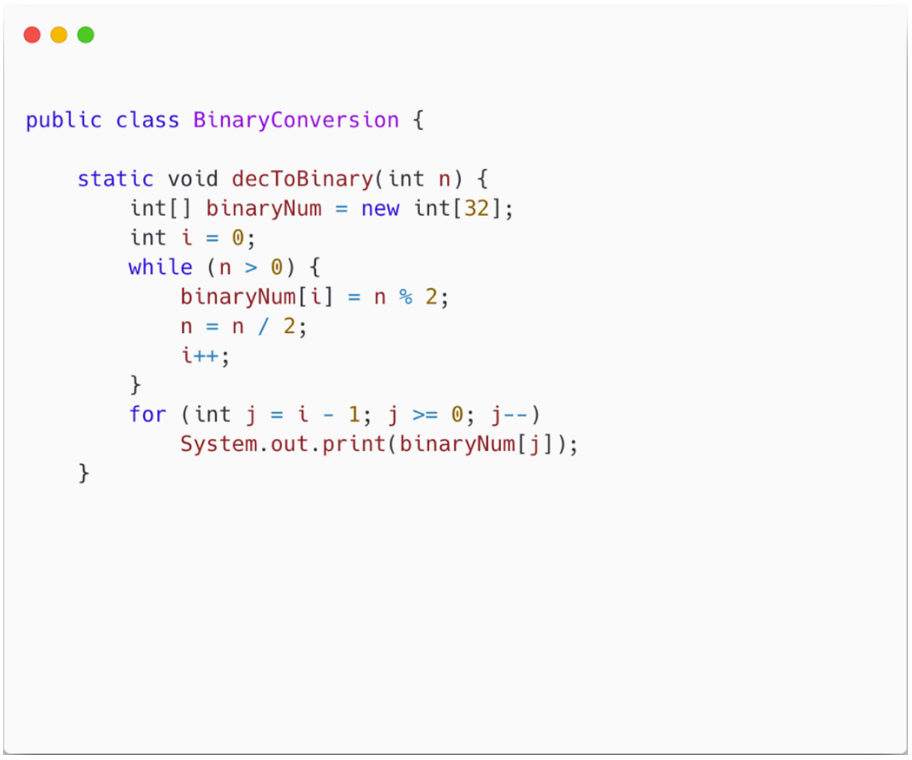


```
public class Power {

    public static int power(int base, int exponent) {
        int result = base;
        for (int i = 2; i <= exponent; i += 1) {
            result = result * base;
        }
        return result;
    }

    public static void main(String[] args) {
        int base = 3;
        int exponent = 3;
        System.out.println(power(base, exponent));
    }
}
```

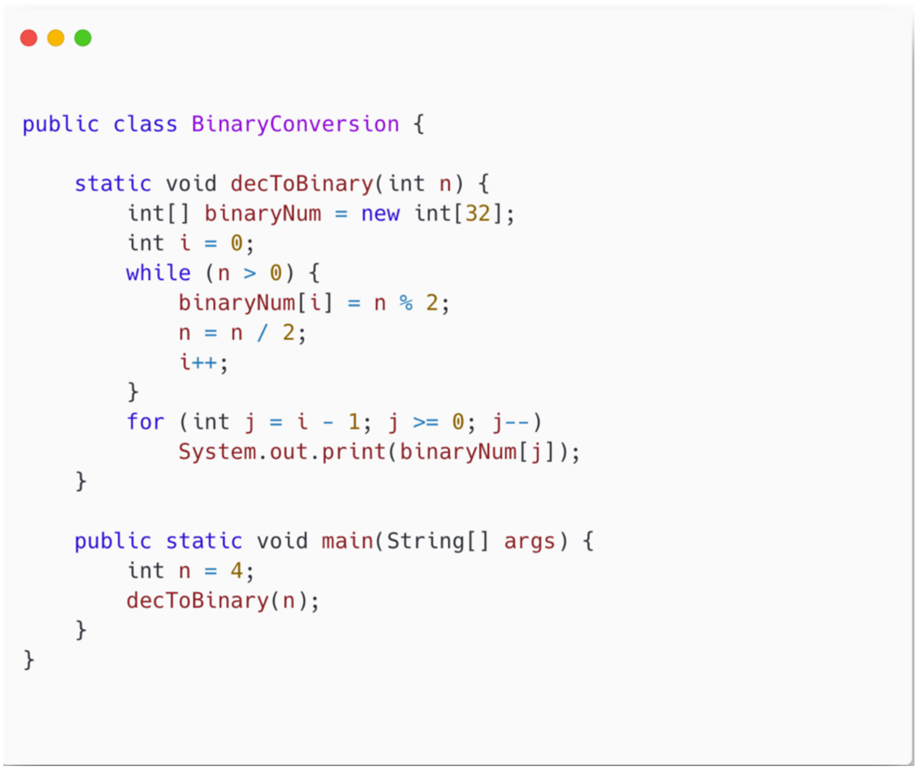
Program 5. Power.



Output screen for Power

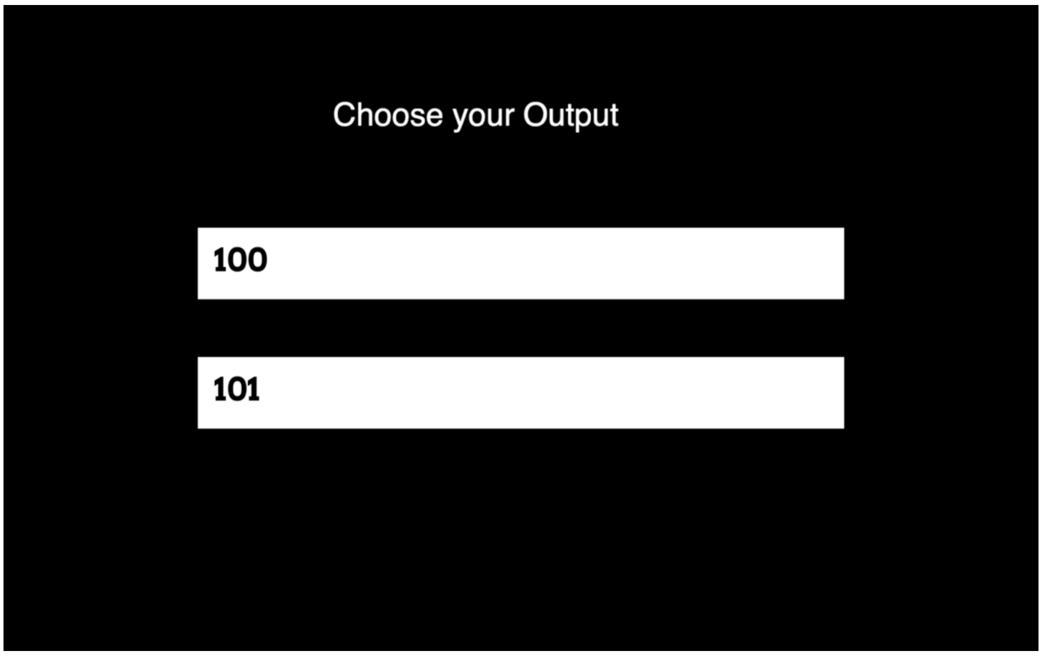


```java
public class BinaryConversion {

    static void decToBinary(int n) {
        int[] binaryNum = new int[32];
        int i = 0;
        while (n > 0) {
            binaryNum[i] = n % 2;
            n = n / 2;
            i++;
        }
        for (int j = i - 1; j >= 0; j--)
            System.out.print(binaryNum[j]);
    }
```

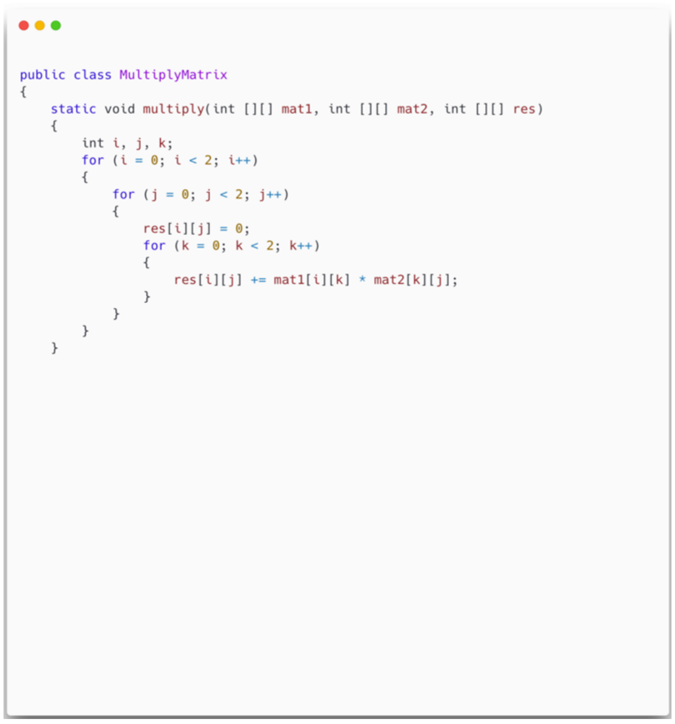


```java
public class BinaryConversion {

    static void decToBinary(int n) {
        int[] binaryNum = new int[32];
        int i = 0;
        while (n > 0) {
            binaryNum[i] = n % 2;
            n = n / 2;
            i++;
        }
        for (int j = i - 1; j >= 0; j--)
            System.out.print(binaryNum[j]);
    }

    public static void main(String[] args) {
        int n = 4;
        decToBinary(n);
    }
}
```

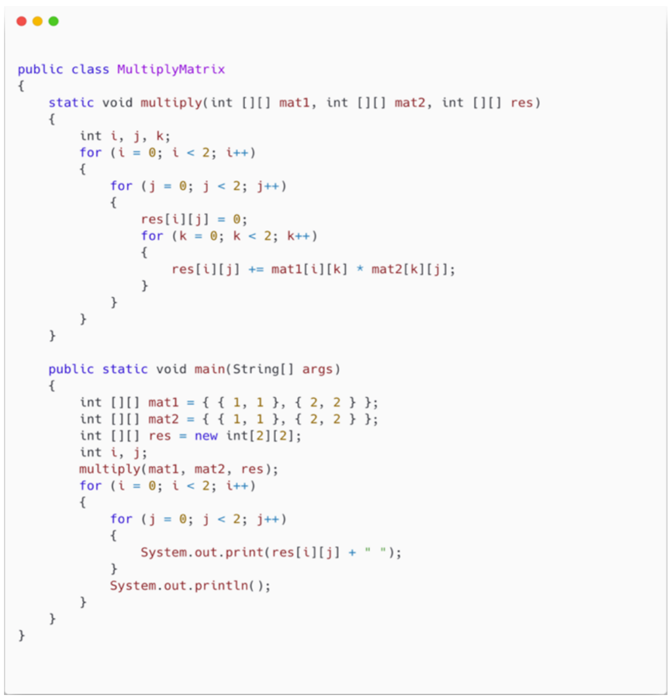
Program 6. Binary Conversion



Output screen for BinaryConversion.

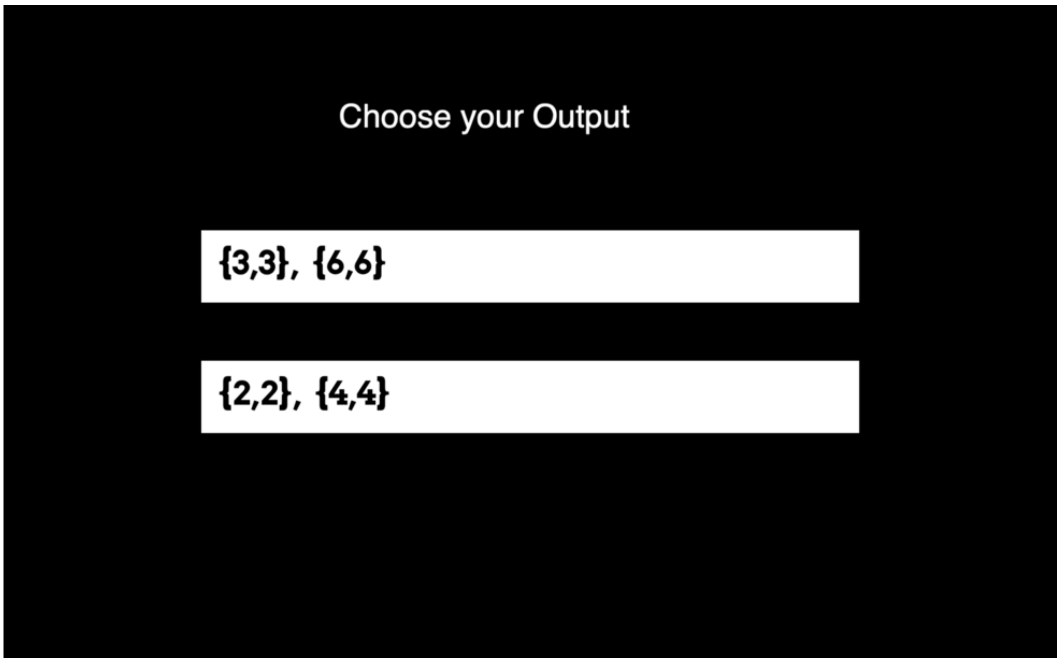


```
public class MultiplyMatrix
{
    static void multiply(int [][] mat1, int [][] mat2, int [][] res)
    {
        int i, j, k;
        for (i = 0; i < 2; i++)
        {
            for (j = 0; j < 2; j++)
            {
                res[i][j] = 0;
                for (k = 0; k < 2; k++)
                {
                    res[i][j] += mat1[i][k] * mat2[k][j];
                }
            }
        }
    }
```



```
public class MultiplyMatrix
{
    static void multiply(int [][] mat1, int [][] mat2, int [][] res)
    {
        int i, j, k;
        for (i = 0; i < 2; i++)
        {
            for (j = 0; j < 2; j++)
            {
                res[i][j] = 0;
                for (k = 0; k < 2; k++)
                {
                    res[i][j] += mat1[i][k] * mat2[k][j];
                }
            }
        }
    }

    public static void main(String[] args)
    {
        int [][] mat1 = { { 1, 1 }, { 2, 2 } };
        int [][] mat2 = { { 1, 1 }, { 2, 2 } };
        int [][] res = new int[2][2];
        int i, j;
        multiply(mat1, mat2, res);
        for (i = 0; i < 2; i++)
        {
            for (j = 0; j < 2; j++)
            {
                res[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

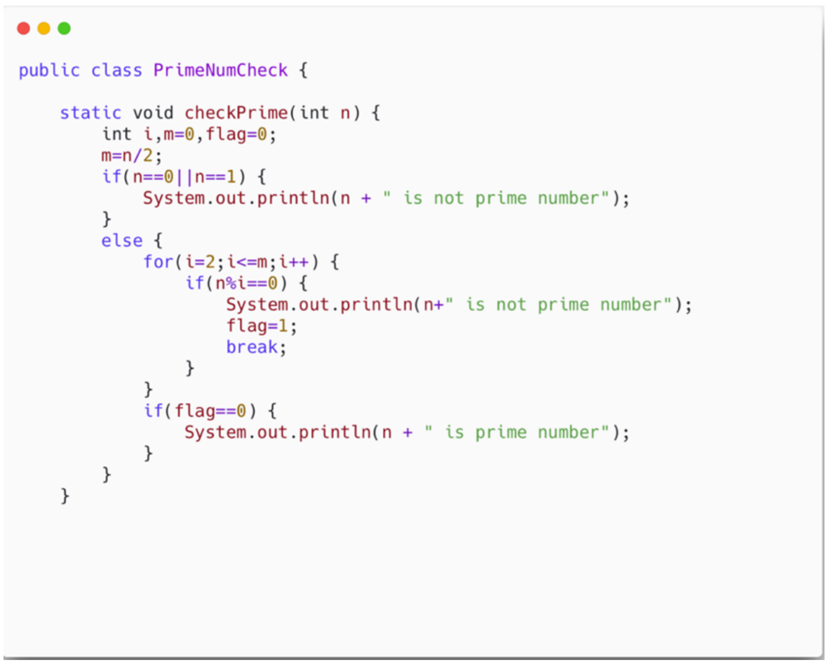
Program 7. MultiplyMatrix


Choose your Output

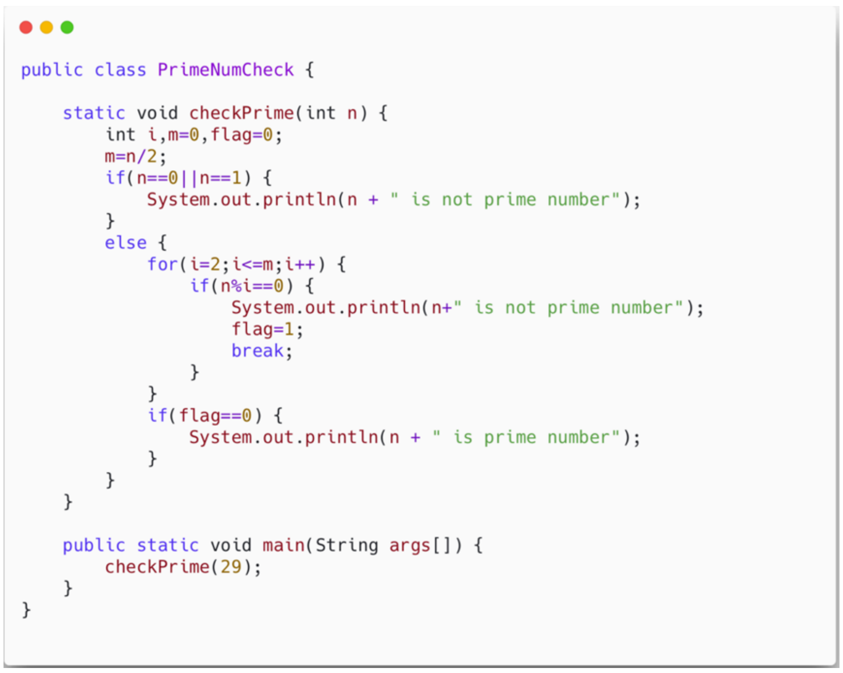
{3,3}, {6,6}

{2,2}, {4,4}

Output Screen for Multipy Matrix

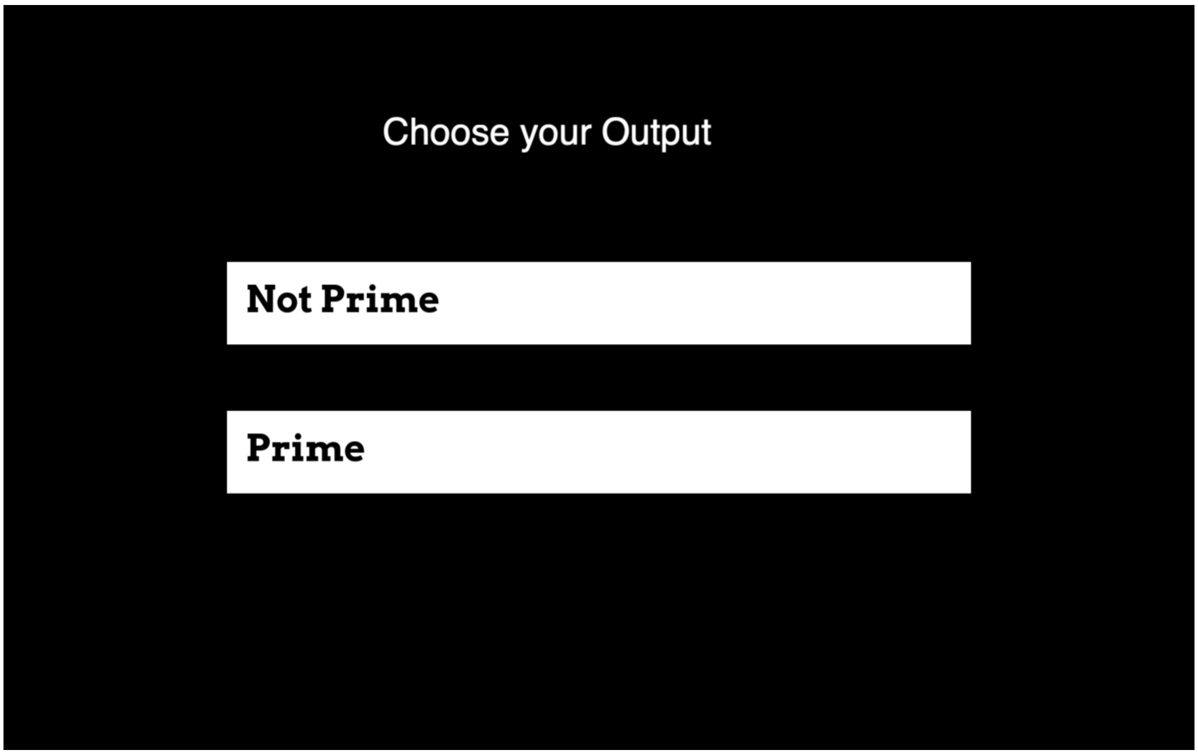


```
public class PrimeNumCheck {

    static void checkPrime(int n) {
        int i,m=0,flag=0;
        m=n/2;
        if(n==0||n==1) {
            System.out.println(n + " is not prime number");
        }
        else {
            for(i=2;i<=m;i++) {
                if(n%i==0) {
                    System.out.println(n+" is not prime number");
                    flag=1;
                    break;
                }
            }
            if(flag==0) {
                System.out.println(n + " is prime number");
            }
        }
    }
```



```
public class PrimeNumCheck {

    static void checkPrime(int n) {
        int i,m=0,flag=0;
        m=n/2;
        if(n==0||n==1) {
            System.out.println(n + " is not prime number");
        }
        else {
            for(i=2;i<=m;i++) {
                if(n%i==0) {
                    System.out.println(n+" is not prime number");
                    flag=1;
                    break;
                }
            }
            if(flag==0) {
                System.out.println(n + " is prime number");
            }
        }
    }

    public static void main(String args[]) {
        checkPrime(29);
    }
}
```

Program 8. Prime Number



Output screen for Prime Number

Thank you for your participation.

Press <spacebar> to exit