Comparative analysis of 6 programming languages based on readability, writability and reliability

Ishrat Jahan Ananya 1631636042

Farsihta Jayas Kinjol 1721411042

Zahin Ahmed 1711020042

Sabbir Hasan 1411107042

Bitan Debnath 1721929042

# Abstract:

# Introduction:

A programming language is a formal language used to communicate with machines such as computers. Programming languages consist of a set of instructions, usually called the “syntax”, and rules on how the syntax can be used in combination with each other. The first programming language “Plankalkül” was designed by German scientist Konrad Zuse, however it was never implemented[1]. Short Code, one of the first pseudocode languages, was developed by John Mauchly in 1949. Soon after, FORTRAN became the first high level programming language that was commercially implemented in 1954 by John Backus[1]. Consequently, many languages have been developed over the years for many different purposes. Some languages have been designed to be suitable for complex computations, while others cater to other domains such as Artificial Intelligence. There are two major categories of programming languages: Imperative languages and Declarative languages. Imperative languages such as C, C++, and Java are more commonly used by general people and are often called general purpose languages as well. Declarative languages such as functional languages (LISP) have specific use cases.

While most imperative programming languages have some basic similarities, the difference usually lies in syntax, how the syntax is converted to machine language, efficiency, closeness to machine language (high level vs low level) etc. In recent years, a lot of emphasis has been given on making languages more readable and writable for programmers. Languages such as Python, R, Julia etc. are much higher level compared to the languages they are built on (mostly C, C++, S and Scheme) [2], [3], and they were developed with the aim of higher abstraction and making the whole process of coding easier and more convenient*//insert python tutorial reference when mendeley is working*. Indeed, the readability and writability of code holds great importance to all types of users, starting from non-programmers, novice programmers or even experienced programmers. In today’s world, almost everyone has to deal with some type of programming or code. People with no experience in coding will not be comfortable in reading or understanding code that is complicated or unable to express its purpose clearly. Similarly, novice programmers will face a huge learning curve if the syntax and rules of a language are not orthogonal and simple[4]. In this case both readability and writability are important, as they are trying to learn the syntax while also implement it themselves. Finally, experienced programmers usually deal with code that is implemented and used in industry. Such codebases are maintained over long periods of time, and the programmer(s) dealing with the code changes often. Hence, not only are readability and writability extremely important here (for the programmers to be able to understand and extend/modify the code as needed), but also reliability comes into play. Reliability of a programming language ensures that the code behaves the way it is supposed to, and carries out its purpose, at all times. This is very important in industry-level programming where the smallest of unforeseen circumstances can cause huge losses.

Hence, the aim of this research is to compare the readability, writability and reliability of 6 commonly used imperative programming languages, which are C, C++, Java, JavaScript, Python and R. All of these programming languages can, and is currently being, used for similar applications hence it is important to compare the differences in readability, writability and reliability of the languages, and the trade-offs that accompany the differences. The study consists of a theoretical comparison between the constructs and design of the programming languages to judge readability, writability and reliability, a survey conducted using code snippets to judge just readability and writability. Finally the readability and writability were compared to the runtimes of two algorithms to analyze whether there’s a trade-off between efficiency and these two factors, and we also tried to check for reliability issues while running these computationally intensive algorithms.

# Literature Review

# Readability, Writability and Reliability

Before embarking on the journey of comparing these three metrics for each language, it is important to understand how each is defined and what they signify.

## Readability

As the term suggests, “readability” of a language is essentially how easy it is to read a piece of code and understand what it is doing [1]. When earlier programming languages were developed, the main factor to be considered was efficiency of the language. However, as the use of these languages increased, it became apparent that if users were taking a long time understanding an existing piece of code, then it was slowing down the process and hence reducing efficiency before the computation even began. Hence, eventually the focus shifted from machine oriented efficiency to human efficiency. Depending on the problem domain, the concept of readability changes. For example, a language that is not meant for complex numerical computations, such as vanilla JavaScript, might not make a very readable script for mathematical computations.

## Writability

The writability of a programming language is defined by the ease of creating a new program for a specific problem domain [1]. Similar to readability, writability of a language is also heavily influenced by the problem domain it is being used in. An easy and common example for this would be the difference in writability for a program with a Graphical User Interface (GUI). A language such as Visual BASIC or Java would have high writability for such programs as they are designed for such applications, whereas C would have very low writability as it is simply not designed for programs that require a GUI.

## Reliability

The performance of a program depends considerably on the language it has been written on. Its affects the decision of whether the program will be reliable for further extension and modification in the future if required. Of course reliability is a more abstract term and not all languages need to have the same standard and it depends on the purpose of the program it’s written with.

A significant criterion to base our judgment has been mentioned in John D Gannon and J.J Horning’s paper “Language Design for Programming Reliability*”//insert ishrat’s first reference in reliability* which is the ability of a programming language to decrease programming errors and the ability to detect them if any. This can be related to the readability and writability of the said programming language. A language that is more readable theoretically should have less fuss to deal with when trying to detect anomalies or errors.

In the table below, factors that affect readability, writability and reliability are shown. Short descriptions of each factor is given afterwards.

Table 1 Factors affecting Readability, Writability and Reliability

|  |  |  |
| --- | --- | --- |
| Readability | Writability | Reliability |
| Simplicity | Simplicity | Simplicity |
| Orthogonality | Orthogonality | Orthogonality |
| Data types | Data types | Data types |
| Syntax design | Syntax design | Syntax design |
|  | Support for abstraction | Support for abstraction |
|  | Expressivity | Expressivity |
|  |  | Type checking |
|  |  | Exception handling |
|  |  | Restricted aliasing |

## Factors

* Simplicity:
* Orthogonality:
* Data types:
* Syntax design:
* Support for abstraction:
* Expressivity:
* Type checking:
* Exception Handling:
* Restricted aliasing:

Aliasing is when two pointers are directing to the same memory location. This is undesirable as we don’t want mismatched data, undefined results or overwrite any value. Thus a strict aliasing rule is often integrated within the language system to prevent pointers of different objects to never indicate to the same memory location.