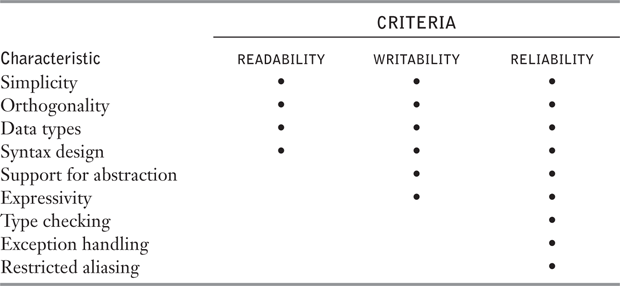
**Part 3 - Language Evaluation Criteria**

**// Directions**

Use the language evaluation criteria listed in our textbook under Chapter 1 Section 1.3 Language Evaluation Criteria to evaluate the three programming languages with respect to solving the above mentioned problem. Tabulate your results and write a brief description explaining your findings.

File to upload: project\_report\_last\_names.odt/docx



**// Evaluations & Findings**

**Language 1: C++**

...

**Language 2: Java**

...

**Language 3: Python**

...

**Below are summarized descriptions of the definitions provided for each criteria point in our textbook.**

**// Readability**

The significance of readability in programming languages became apparent in the 1970s, marking a shift from a focus on machine efficiency to human understanding. Before this era, coding was prioritized over maintenance, but as software life-cycle concepts evolved, readability gained prominence. The quality of programs and programming languages began to be measured by their ease of maintenance, recognizing the impact of readability. Consideration of readability must align with the problem domain, as programs written in unsuitable languages may become convoluted and challenging to understand. Subsequent sections delve into specific characteristics contributing to the readability of programming languages.

**// Overall Simplicity**

The overall simplicity of a programming language significantly impacts its readability, with a large number of constructs making it harder to learn. Programmers often focus on a subset of a complex language, leading to readability issues when their subset differs from that of the reader. Feature multiplicity, exemplified by multiple ways to perform an operation, and operator overloading, where a symbol has multiple meanings, can further complicate understanding. While simplicity is crucial, an extreme level, as seen in assembly language, can hinder readability due to the lack of complex control structures and increased code volume. Striking the right balance is essential to maintain readability without sacrificing functionality.

**// Orthogonality**

Orthogonality in programming languages refers to the ability to combine a small set of primitive constructs in various ways to build control and data structures. A language is considered orthogonal when every possible combination of primitives is legal and meaningful, resulting in fewer exceptions and greater regularity in design. The concept enhances simplicity and readability, but excessive orthogonality can lead to unnecessary complexity. Examples from languages like C illustrate context-dependent rules and exceptions, highlighting the challenge of achieving the right balance. Functional languages, such as Lisp, are seen as offering simplicity and orthogonality, but factors like efficiency have limited their widespread adoption.

**// Data Types**

The presence of adequate facilities for defining data types and data structures in a language is another significant aid to readability. For example, suppose a numeric type is used for an indicator flag because there is no Boolean type in the language. In such a language, for example, in the original version of C, we might have an assignment such as the following: “timeout = 1”

The meaning of this statement is unclear, whereas in a language that includes Boolean types, we would have the following: “timeout = **true”** The meaning of this statement is perfectly clear.

**// Syntax Design**

The syntax design of a programming language significantly impacts program readability. For instance, the choice of special words and their use as variable names can influence program appearance and comprehension. Some languages, like C, use braces to form compound statements, which may reduce readability due to the uniformity of statement group termination. Conversely, languages like Fortran 95 and Ada enhance clarity by employing distinct closing syntax for different statement groups. The relationship between form and meaning is crucial, and violations of this principle, such as similar-looking constructs with different meanings in C, can hinder readability. The text emphasizes the importance of designing statements to visually convey their purpose for improved program understanding.

**// Writeability**

Writability in programming languages measures the ease of creating programs for a specific problem domain. Many factors influencing readability also impact writability, as writing a program involves frequent re-reading of existing code. Comparing the writability of two languages should consider their target problem domains, as languages designed for specific applications may excel in those contexts. For instance, Visual BASIC (VB) and C exhibit significantly different writabilities for tasks like creating a graphical user interface (GUI) or writing systems programs, reflecting their respective design focuses. The subsequent subsections outline key characteristics that influence the writability of a language.

**// Expressivity**

**// Abstraction**

**// Type Cheching**

**// Language Evaluation** Most criteria, particularly readability, writability, and reliability, are neither precisely defined nor accurately measurable. Nevertheless, they are useful concepts and they provide valuable insight into the design and evaluation of programming languages.

A final note on evaluation criteria: language design criteria are weighed differently from different perspectives. Language implementers are concerned primarily with the difficulty of implementing the constructs and features of the language. Language users are worried about writability first and readability later. Language designers are likely to emphasize elegance and the ability to attract widespread use. These characteristics often conflict with one another.