Unreadable code often leads to bugs, inefficiencies, and duplicated code.  
To produce machine code, the source code must either be compiled or transpiled.  
However, with the concept of the stored-program computer introduced in 1949, both programs and data were stored and manipulated in the same way in computer memory.  
It affects the aspects of quality above, including portability, usability and most importantly maintainability.  
Scripting and breakpointing is also part of this process.  
Provided the functions in a library follow the appropriate run-time conventions (e.g., method of passing arguments), then these functions may be written in any other language.  
Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly.  
For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input.  
A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it.  
Transpiling on the other hand, takes the source-code from a high-level programming language and converts it into bytecode.  
Also, specific user environment and usage history can make it difficult to reproduce the problem.  
However, because an assembly language is little more than a different notation for a machine language, two machines with different instruction sets also have different assembly languages.  
 Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.  
Ideally, the programming language best suited for the task at hand will be selected.  
There exist a lot of different approaches for each of those tasks.