In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages.  
They are the building blocks for all software, from the simplest applications to the most sophisticated ones.  
Assembly languages were soon developed that let the programmer specify instruction in a text format (e.g., ADD X, TOTAL), with abbreviations for each operation code and meaningful names for specifying addresses.  
Many applications use a mix of several languages in their construction and use.  
Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute.  
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.  
Proficient programming thus usually requires expertise in several different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.  
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 example, when a bug in a compiler can make it crash when parsing some large source file, a simplification of the test case that results in only few lines from the original source file can be sufficient to reproduce the same crash.  
 These compiled languages allow the programmer to write programs in terms that are syntactically richer, and more capable of abstracting the code, making it easy to target varying machine instruction sets via compilation declarations and heuristics.  
Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment.  
There exist a lot of different approaches for each of those tasks.  
However, readability is more than just programming style.  
 Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.  
 Various visual programming languages have also been developed with the intent to resolve readability concerns by adopting non-traditional approaches to code structure and display.