Programmers typically use high-level programming languages that are more easily intelligible to humans than machine code, which is directly executed by the central processing unit. Some of these factors include: The presentation aspects of this (such as indents, line breaks, color highlighting, and so on) are often handled by the source code editor, but the content aspects reflect the programmer's talent and skills. Programmable devices have existed for centuries. FORTRAN, the first widely used high-level language to have a functional implementation, came out in 1957, and many other languages were soon developed—in particular, COBOL aimed at commercial data processing, and Lisp for computer research. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. Many applications use a mix of several languages in their construction and use. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). The academic field and the engineering practice of computer programming are both largely concerned with discovering and implementing the most efficient algorithms for a given class of problems. 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. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. Many programmers use forms of Agile software development where the various stages of formal software development are more integrated together into short cycles that take a few weeks rather than years. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. 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. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Computer programming or coding is the composition of sequences of instructions, called programs, that computers can follow to perform tasks. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Programmable devices have existed for centuries. However, readability is more than just programming style. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances. 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. For example, COBOL is still strong in corporate data centers often on large mainframe computers, Fortran in engineering applications, scripting languages in Web development, and C in embedded software. For example, COBOL is still strong in corporate data centers often on large mainframe computers, Fortran in engineering applications, scripting languages in Web development, and C in embedded software. Expert programmers are familiar with a variety of well-established algorithms and their respective complexities and use this knowledge to choose algorithms that are best suited to the circumstances.