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. Ideally, the programming language best suited for the task at hand will be selected. 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. Auxiliary tasks accompanying and related to programming include analyzing requirements, testing, debugging (investigating and fixing problems), implementation of build systems, and management of derived artifacts, such as programs' machine code. Trial-and-error/divide-and-conquer is needed: the programmer will try to remove some parts of the original test case and check if the problem still exists. Their jobs usually involve: Although programming has been presented in the media as a somewhat mathematical subject, some research shows that good programmers have strong skills in natural human languages, and that learning to code is similar to learning a foreign language. 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. High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware. Programming languages are essential for software development. Methods of measuring programming language popularity include: counting the number of job advertisements that mention the language, the number of books sold and courses teaching the language (this overestimates the importance of newer languages), and estimates of the number of existing lines of code written in the language (this underestimates the number of users of business languages such as COBOL). 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. 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. The first compiler related tool, the A-0 System, was developed in 1952 by Grace Hopper, who also coined the term 'compiler'. Integrated development environments (IDEs) aim to integrate all such help. Many applications use a mix of several languages in their construction and use. Code-breaking algorithms have also existed for centuries. Later a control panel (plug board) added to his 1906 Type I Tabulator allowed it to be programmed for different jobs, and by the late 1940s, unit record equipment such as the IBM 602 and IBM 604, were programmed by control panels in a similar way, as were the first electronic computers. Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability. Different programming languages support different styles of programming (called programming paradigms). Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. New languages are generally designed around the syntax of a prior language with new functionality added, (for example C++ adds object-orientation to C, and Java adds memory management and bytecode to C++, but as a result, loses efficiency and the ability for low-level manipulation). In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form. In 1206, the Arab engineer Al-Jazari invented a programmable drum machine where a musical mechanical automaton could be made to play different rhythms and drum patterns, via pegs and cams. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. 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.