Allen Downey, in his book How To Think Like A Computer Scientist, writes: Many computer languages provide a mechanism to call functions provided by shared libraries. 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. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. 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. 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. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. In the 1880s, Herman Hollerith invented the concept of storing data in machine-readable form. They are the building blocks for all software, from the simplest applications to the most sophisticated ones. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. While these are sometimes considered programming, often the term software development is used for this larger overall process – with the terms programming, implementation, and coding reserved for the writing and editing of code per se. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. The choice of language used is subject to many considerations, such as company policy, suitability to task, availability of third-party packages, or individual preference. Different programming languages support different styles of programming (called programming paradigms). Unreadable code often leads to bugs, inefficiencies, and duplicated code. Techniques like Code refactoring can enhance readability. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. 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. In the 9th century, the Arab mathematician Al-Kindi described a cryptographic algorithm for deciphering encrypted code, in A Manuscript on Deciphering Cryptographic Messages. 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. In 1801, the Jacquard loom could produce entirely different weaves by changing the "program" – a series of pasteboard cards with holes punched in them. Integrated development environments (IDEs) aim to integrate all such help. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm. He gave the first description of cryptanalysis by frequency analysis, the earliest code-breaking algorithm.