They are the building blocks for all software, from the simplest applications to the most sophisticated ones. 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. The first step in most formal software development processes is requirements analysis, followed by testing to determine value modeling, implementation, and failure elimination (debugging). Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation. It affects the aspects of quality above, including portability, usability and most importantly maintainability. However, Charles Babbage had already written his first program for the Analytical Engine in 1837. Normally the first step in debugging is to attempt to reproduce the problem. 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. 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. 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. Use of a static code analysis tool can help detect some possible problems. Integrated development environments (IDEs) aim to integrate all such help. However, readability is more than just programming style. 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. Proficient programming usually requires expertise in several different subjects, including knowledge of the application domain, details of programming languages and generic code libraries, specialized algorithms, and formal logic. Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation. Ideally, the programming language best suited for the task at hand will be selected. 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. It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages. 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. Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards. 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. Many applications use a mix of several languages in their construction and use. Programs were mostly entered using punched cards or paper tape. 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.