They are the building blocks for all software, from the simplest applications to the most sophisticated ones. 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. 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). Readability is important because programmers spend the majority of their time reading, trying to understand, reusing and modifying existing source code, rather than writing new source code. Different programming languages support different styles of programming (called programming paradigms). Some languages are more prone to some kinds of faults because their specification does not require compilers to perform as much checking as other languages. A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it. 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). 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. Whatever the approach to development may be, the final program must satisfy some fundamental properties. A similar technique used for database design is Entity-Relationship Modeling (ER Modeling). 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). When debugging the problem in a GUI, the programmer can try to skip some user interaction from the original problem description and check if remaining actions are sufficient for bugs to appear. Readability is important because programmers spend the majority of their time reading, trying to understand, reusing and modifying existing source code, rather than writing new source code. 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. 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. It is usually easier to code in "high-level" languages than in "low-level" ones. 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. Different programming languages support different styles of programming (called programming paradigms). Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages. This can be a non-trivial task, for example as with parallel processes or some unusual software bugs. Use of a static code analysis tool can help detect some possible problems. 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. 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. Code-breaking algorithms have also existed for centuries.