The purpose of programming is to find a sequence of instructions that will automate the performance of a task (which can be as complex as an operating system) on a computer, often for solving a given problem.  
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.  
Relatedly, software engineering combines engineering techniques and principles with 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).  
However, Charles Babbage had already written his first program for the Analytical Engine in 1837.  
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.  
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.  
Integrated development environments (IDEs) aim to integrate all such help.  
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.  
Use of a static code analysis tool can help detect some possible problems.  
Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment.  
One approach popular for requirements analysis is Use Case analysis.  
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.  
Also, specific user environment and usage history can make it difficult to reproduce the problem.