Integrated development environments (IDEs) aim to integrate all such help..  
It is usually easier to code in "high-level" languages than in "low-level" ones.  
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.  
Trade-offs from this ideal involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency with which programs written in a given language execute.  
Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment.  
Use of a static code analysis tool can help detect some possible problems.  
 It is very difficult to determine what are the most popular modern programming languages.  
Unreadable code often leads to bugs, inefficiencies, and duplicated code.  
 Programs were mostly entered using punched cards or paper tape.  
One approach popular for requirements analysis is Use Case analysis.  
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.  
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.  
In 1801, the Jacquard loom could produce entirely different weaves by changing the "program" – a series of pasteboard cards with holes punched in them.  
 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.  
It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages.