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.  
For this purpose, algorithms are classified into orders using so-called Big O notation, which expresses resource use, such as execution time or memory consumption, in terms of the size of an input.  
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.  
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.  
To produce machine code, the source code must either be compiled or transpiled.  
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.  
Techniques like Code refactoring can enhance readability.  
Scripting and breakpointing is also part of this process.  
A study found that a few simple readability transformations made code shorter and drastically reduced the time to understand it.  
Integrated development environments (IDEs) aim to integrate all such help.  
As early as the 9th century, a programmable music sequencer was invented by the Persian Banu Musa brothers, who described an automated mechanical flute player in the Book of Ingenious Devices.  
There exist a lot of different approaches for each of those tasks.  
Compiling takes the source code from a low-level programming language and converts it into machine code.  
 After the bug is reproduced, the input of the program may need to be simplified to make it easier to debug.  
Use of a static code analysis tool can help detect some possible problems.