Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages..  
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.  
Integrated development environments (IDEs) aim to integrate all such help.  
Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability.  
 Following a consistent programming style often helps readability.  
However, readability is more than just programming style.  
Unreadable code often leads to bugs, inefficiencies, and duplicated code.  
By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers.  
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.  
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.  
 High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware.  
 The first computer program is generally dated to 1843, when mathematician Ada Lovelace published an algorithm to calculate a sequence of Bernoulli numbers, intended to be carried out by Charles Babbage's Analytical Engine.  
Normally the first step in debugging is to attempt to reproduce the problem.  
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.  
 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.