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.  
 Implementation techniques include imperative languages (object-oriented or procedural), functional languages, and logic languages.  
Text editors were also developed that allowed changes and corrections to be made much more easily than with punched cards.  
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.  
 Allen Downey, in his book How To Think Like A Computer Scientist, writes:  
 Many computer languages provide a mechanism to call functions provided by shared libraries.  
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.  
Provided the functions in a library follow the appropriate run-time conventions (e.g., method of passing arguments), then these functions may be written in any other language.  
 Programs were mostly entered using punched cards or paper tape.  
Some of these factors include:  
 The presentation aspects of this (such as indents, line breaks, color highlighting, and so on) are often handled by the source code editor, but the content aspects reflect the programmer's talent and skills.  
 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.  
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.  
Some text editors such as Emacs allow GDB to be invoked through them, to provide a visual environment.  
 Various visual programming languages have also been developed with the intent to resolve readability concerns by adopting non-traditional approaches to code structure and display.  
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.