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.  
 Debugging is often done with IDEs. Standalone debuggers like GDB are also used, and these often provide less of a visual environment, usually using a command line.  
While these are sometimes considered programming, often the term software development is used for this larger overall process – with the terms programming, implementation, and coding reserved for the writing and editing of code per se.  
However, with the concept of the stored-program computer introduced in 1949, both programs and data were stored and manipulated in the same way in computer memory.  
Many factors, having little or nothing to do with the ability of the computer to efficiently compile and execute the code, contribute to readability.  
By the late 1960s, data storage devices and computer terminals became inexpensive enough that programs could be created by typing directly into the computers.  
 Popular modeling techniques include Object-Oriented Analysis and Design (OOAD) and Model-Driven Architecture (MDA).  
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.  
 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.  
It involves designing and implementing algorithms, step-by-step specifications of procedures, by writing code in one or more programming languages.  
Unreadable code often leads to bugs, inefficiencies, and duplicated code.  
Compilers harnessed the power of computers to make programming easier by allowing programmers to specify calculations by entering a formula using infix notation.  
 Machine code was the language of early programs, written in the instruction set of the particular machine, often in binary notation.  
 High-level languages made the process of developing a program simpler and more understandable, and less bound to the underlying hardware.  
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.