Computer programming

Computer programming (often simply **programming** or **coding**) is the craft of writing a **set of commands or instructions** that can later be **compiled** and/or **interpreted** and then inherently transformed to an **executable** that an electronic machine can execute or "run". Programming requires mainly logic, but has elements of science, mathematics, engineering, and — many would argue — art.

In software engineering, programming (*implementation*) is regarded as one phase in a **software development process**.

Programming languages

The **programming language** a computer can directly execute is **machine language** (sometimes called "**machine code**"). Originally all **programmers** worked out every detail of the machine code, but this is hardly ever done anymore. Instead, programmers write **source code**, and a computer (running a **compiler**, an **interpreter** or occasionally an **assembler**) **translates** it through one or more **translation steps** to fill in all the details, before the final machine code is executed on the **target computer**. Even when complete **low-level control** of the target computer is required, programmers write **assembly language,** whose instructions are **mnemonic** one-to-one transcriptions of the corresponding machine language instructions.

Different programming languages support different styles of programming (called **programming paradigms**). Part of the art of programming is selecting one of the programming languages best suited for the task at hand. Different programming languages require different levels of detail to be handled by the programmer when implementing algorithms, often in a compromise between ease of use and performance (a trade-off between "programmer time" and "computer time").

In some languages, an interpretable **p-code binary** (or **byte-code**) is generated, rather than machine language. **Bytecode** is used in the popular **Java programming language** by Sun Microsystems as well as Microsoft's recent **.NET** family of languages (MS.NET's P-Code is called the **Intermediate Language** or **IL**) and **Visual Basic** previous to the .NET version.

A brief history of programming

The earliest programmable machine (that is, a machine that can adjust its capabilities based upon changing its "program") can be said to be the Jacquard Loom, which was developed in 1801. The machine used a series of **pasteboard cards** with holes **punched** in them. The **hole pattern** represented the pattern that the loom had to follow in weaving cloth. The loom could produce entirely different weaves using different sets of cards. This innovation was later refined by Herman Hollerith of IBM in the development of the famous **IBM punch card.**

Another early use of computer programs was made using a **soldering iron** and a large number of **vacuum tubes** (later transistors). As programs became more complex, this became almost impossible, as one mistake would likely render the whole program useless. As **data storage media** became more advanced, it became possible to re-use one program for many things according to the content of the memory. A person would spend quite some time making punch cards that would hold a list of instructions for a computer. Every model of computer would be likely to need different instructions to do the same task. As computers became more **powerful**, and storage media became re-usable, it became possible to use the computer to make the program. Programmers quickly began to prefer text over 1s and 0s, and punch cards were phased out. As time has progressed computers have made giant leaps in the area of processing power. This has brought about newer programing languages that are more **abstracted** from the **underlying hardware**. Although these more abstracted languages require additional **overhead**, in most cases the huge increase in speed of modern computers has brought about little performance decrease compared to earlier counterparts. The benefits of these more abstracted languages is that they allow both an easier learning curve for people less familiar with the older lower-level programming languages, and they also allow a more experienced programmer to develop simple applications quickly. Despite these benefits, large **complicated programs**, and programs that are more dependent on speed still require the faster and relatively **lower-level languages** with todays hardware.

Some forms of programming have been increasingly subject to offshore outsourcing (importing software and services from other countries, usually lower-wage), making programming career decisions more complicated. It is unclear how far this trend will continue and how deeply it will impact programmer wages and opportunities.

Examples of computer programming languages

- **C** is a **compiled procedural, imperative** programming language made popular as the basis of Unix.
- **C++** is a compiled programming language based on C, with support for **object-oriented programming.** It is one of the most widely-used programming languages currently available. It is often considered to be the **industry-standard language** of **game development,** but is also very often used to write other types of computer **software applications.** C++ was developed by Bjarne Stroustrup and was based on the programming language C. C++ retains the **syntax** and many familiar functions of C, but also adds various concepts associated with other **programming paradigms**, such as **classes**.
- **C#** is an object-oriented programming language developed by Microsoft as part of their .NET initiative, and later approved as a standard by ECMA and ISO. C# has a procedural, object oriented syntax based on C++ that includes aspects of several other programming languages (most notably Delphi, Visual Basic, and Java) with a particular emphasis on simplification (less symbolic requirements than C++, less decorative requirements than Java).
- **FORTRAN** is a **general-purpose**, procedural, imperative programming language that is especially suited to numeric computation and **scientific computing**. Originally developed by International Business Machines Corporation (IBM) in the 1950s for scientific and engineering applications.
- **Java** is an object oriented interpreted programming language. It has gained popularity in the past few years for its ability to be run on many **platforms**, including Microsoft Windows, Linux, Mac OS, and other systems. It was developed by Sun Microsystems.
- **Lisp** is a family of functional, sometimes **scripted**, programming languages often used in artificial intelligence (or **AI)**.
- **Pascal** is a general-purpose structured language named after the famous mathematician and philosopher Blaise Pascal. It was very popular during the 80's and

90's. Whilst popularity of Pascal itself has waned (its principal use is in teaching of programming) languages derived from it (such as Borland Delphi) are still in use.

- **BASIC** (Beginner's All purpose Symbolic Instruction Code) was mostly used when microcomputers first hit the market, in the 70's and 80's, but was largely replaced by other languages such as C, Pascal and Java. Whilst some commercial applications have been written in BASIC it has typically been seen as a language for learning/teaching programming rather than as a language for serious development. Different **implementations** varied widely in the functionality they offered, most lacked important features such as strong data typing, procedures and functions.
- **Visual Basic** designed and developed by Microsoft, is integrated into a visual development **interface.**
- **PHP** is a newer programming language with focus on **web design** and a C-like syntax.
- **Shell scripting**, in particular using either a variant of the Bourne shell or the C shell, is popular among UNIX **hackers**. Altough the exact implementation varies among different shells, the core principles remain intact: only providing facilities for program flow (also seen in C) while placing emphasis on using external programs, although most shells feature some other functions internally, known as **builtins**. Shell **scripting** is used primarly for **Casual programming**, ranging from user-defined functions to complete programs, and everything in between. It is also used as a tool for rapid **prototyping** when the exact design of a program is not yet clear enough for a full implementation, often in a **Compiled language** like C.