Computer programming

Today, most people don't need to know how a computer works. Most people can simply turn on a computer or a mobile phone and point at some little graphical object on the display, click a button or swipe a finger or two, and the computer does something. An example would be to get weather information from the net and display it. How to interact with a computer program is all the average person needs to know. But, since you are going to learn how to write computer programs, you need to know a little bit about how a computer works. Your job will be to instruct the computer to do the following things:

1. **Process**: A series of actions or steps taken to achieve an end.
2. **Procedure**: A series of actions conducted in a certain order.
3. **Algorithm**: An ordered set of steps to solve a problem.

Basically, writing *software* (computer programs) involves describing *processes*, *procedures*; it involves the authoring of *algorithms*. Computer programming involves developing lists of instructions - the source code representation of software The stuff that these instructions manipulate are different types of objects, e.g., numbers, words, images, sounds, etc... Creating a computer program can be like composing music, like designing a house, like creating lots of stuff. It has been argued that in its current state it is an *art*, not engineering.

Computer programming (often shortened to programming or coding) is the process of writing, testing, and maintaining the source code of computer programs. The source code is written in a programming language. This code may be a modification of the existing source or something completely new. The process of writing a source code requires expertise in many different subjects, including the knowledge of the application domain and algorithms. Within software engineering, programming is regarded as one phase in a software development process.

The final program produced by computer programmers must satisfy some fundamental properties. The following five properties are among the most relevant:

Efficiency/performance: The fewer amounts of system resources the program consumes, the better. This also refers to correct disposal of some resources, such as cleaning up temporary files and lack of memory leaks.

Reliability: How often the results of a program are correct. This depends on conceptual correctness of algorithms, and minimization of programming mistakes, such as mistakes in resource management and logic errors.

Robustness: How well a program anticipates problems not due to a programmer error. This includes situations such as incorrect, inappropriate or corrupt data, unavailability of needed resources such as memory, operating system services and network connections, and user error.

Usability: The ease with which a person can use the program for its intended purpose or in some cases even unanticipated purposes. Such issues can make or break its success even regardless of other issues.

Portability: The range of computer hardware and operating system platforms on which the source code of a program can be compiled/interpreted and run.

An important reason to consider learning about how to program a computer is that the concepts underlying this will be valuable to you, regardless of whether or not you go on to make a career out of it. One thing that you will learn quickly is that a computer is very dumb, but obedient. It does exactly what you tell it to do, which is not necessarily what you wanted. Programming will help you learn the importance of clarity of expression. But, most of all, it can be lots of fun! An associate once said to me "I can't believe I'm paid so well for something I love to do."

Programming Languages

Different programming languages support different styles of programming. The choice of language used may be an individual's choice or may be dictated by a company's policy. Ideally, the programming language best suited for the task at hand will be selected. Trade-offs from this ideal language involve finding enough programmers who know the language to build a team, the availability of compilers for that language, and the efficiency of the programs written in the given language.

High-Level Languages

Almost the entire computer programming these days is done with *high-level* programming languages. There are lots of them and some are quite old. COBOL, FORTRAN, and Lisp were devised in the 1950s!!! As you will see, high-level languages make it easier to describe the pieces of the program you are creating. They help by letting you concentrate on what you are trying to do rather than on how you represent it in specific computer architecture. They abstract away the specifics of the microprocessor in your computer. And, all high-level languages come with large sets of common stuff you need to do, called libraries.

Let's have a look at two computer programming languages: Logo and Java. Logo comes from Bolt, Beranek&Newman (BBN) and Massachusetts Institute of Technology (MIT). Seymour Papert, a scientist at MIT's Artificial Intelligence Laboratory, and co-workers championed this computer programming language in the 70s. More research of its use in educational settings exists than for any other programming language. In fact, the fairly new Scratch Programming Environment (also from MIT) consists of a modern graphical user interface on top of Logo-like functionality.

Java is a fairly recent programming language. It appeared in 1995 just as the Internet was starting to get lots of attention. Java was invented by James Gosling, working at Sun Microsystems. It's sort-of a medium-level language. One of the big advantages of learning Java is that there is a lot of software already written, which will help you write graphical programs that run on the Internet. You get to take advantage of software that thousands of programmers have already written. Java is used in a variety of applications, from mobile phones to massive Internet data manipulation. You get to work with window objects, Internet connection objects, database access objects and thousands of others. Java is the language used to write Android apps. Both Logo and Java have the same sort of stuff needed to write computer programs. Each has the ability to manipulate objects (for example, arithmetic functions for working with numbers). Each lets you compare objects and do a variety of things depending on the outcome of the comparison. Most importantly, they let you define *named* procedures. Named procedures are lists of built-in instructions and other named procedures. The abstraction of naming stuff lets you write programs in a language you yourself define.

Assembler Language

One abstract layer above a computer's native language is assembler language. In assembler language, everything is given human-friendly symbolic names. The programmer works with operations that the microprocessor knows how to do, they have symbolic names. The microprocessor's registers and addresses in the computer's memory are also given meaningful names by the programmer. This is actually a very big step over what a computer understands, but still tedious for writing a large program. Assembler language instructions still have a place for little bits of software that need to interact directly with the microprocessor and/or those that are executed many times.

The Microprocessor's Language

So, all a computer has in it is bits. You've seen how they are used to represent stuff, pixels, numbers and characters. I've mentioned that computers perform operations on the bits, like move them around, add pairs of them together, etc... One final obvious question is: how are instructions that a computer performs represented? Well, if you instructed a computer in its native language (machine language), you would have to write instructions in the form of (yes, once again) binary numbers. This is very, very hard to do. Although the pioneers of computer science did this, no one does this these days.