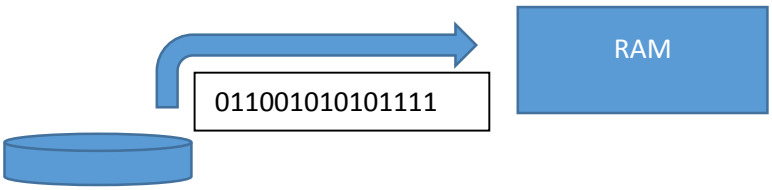
Preliminaries

Before programming it is a good idea to have some idea how a computer works.

A computer, typically, stores its programs and files on a hard disk as magnetic bits. These bits are represented as 0 and 1 and form a binary code. When you start a program, the computer locates the bits belonging to that program and copies how ever many bits are needed to get the programming running into Random Access Memory or RAM. RAM is a set of usually 2 to 4 circuit chips in your computer that can store information temporarily. The reason it does this is to enhance speed and performance. Hard disks spin very fast and locating and reading information can be done in milliseconds, but that is still relatively slow compared to RAM where instructions can be read at nearly the speed of light. Ram is divided up into *addresses.* Variables and commands are stored in these addresses. As elements of program are used most languages release the memory addresses so that other processes and programs can use them.

> Some newer computers have flash drives instead of or in addition to hard drives. Flash drives are similar to RAM but can retain their information and are much quicker than hard drives. The trouble with flash drives at this point is they are expensive and it can be impossible to recover data from them if the drive fails.

From RAM then instructions are passed to the Central Processing Unit (CPU) where they are queued and executed.



A CPU can do just a few things:

- It can fetch instruction
- It can store information.
- It can compare information to check if it is the same, larger or smaller.
- It can copy information from one location to another.
- It can add.

The last bullet is interesting. A computer does all its math by adding binary numbers. To multiply it just adds the number for as many times as the multiplier says. If you multiply 6 by 12, it just adds 6 twelve times. For subtraction it just makes the number to be subtracted a negative number and adds it. Division is more complicated.

Here is some pseudo code for the process:

Set quotient to 0

Align leftmost digits in dividend and divisor
Repeat
    If the dividend bits above the divisor are greater than or equal to the divisor
        Then subtract divisor from that part of the dividend and
        Concatenate 1 to the right hand end of the quotient
        Else concatenate 0 to the right hand end of the quotient
    Shift the divisor one place right
Until the dividend is less than the divisor
quotient is correct, dividend is remainder
STOP

You can see a fairly simple discussion of it at http://www.wikihow.com/Divide-Binary-Numbers

The idea of Pseudo code is to write out the logic of a procedure in a concise but fairly normal English. It is independent of any particular programming language but can easily be applied to any.

Computer processors are really not that smart. They do many of the things they do in the most awkward way imaginable. But, they do it fast, so fast that we don't see the awkwardness. Computers can only do what they are instructed to do. They cannot anticipate of make any leaps of logic. This is what makes programming both so difficult and interesting. You are limited by what the processer can do and you must be explicit about every step to be executed. You cannot leave anything out.

Programming

When you program, you are writing instructions for the computer to execute. Luckily we don't have to write all the instructions. It can take dozens or even hundreds of instructions to do something a simple as writing a statement on a console screen. Modern programming languages such as C# have developed so that one C# command can represent dozens of machine level commands. Modern programming languages offer a bridge between machine language (0s and 1s) and human languages.

Interestingly, almost all programming languages are in English.

Programming languages limit the number of words or terms available for use. They ensure that each term has one and only one meaning in a given context. You can see a complete list of C# keywords at http://msdn.microsoft.com/en-us/library/x53a06bb.aspx .

Programming languages also define a strict syntax that defines a structure for using the terms and defining commands. Most languages are case sensitive, use curly braces to define blocks of code and end statements with semi colons.

It is important to note that computers don't actually understand programming languages. The key words and commands of the programming language must be converted into machine language before the

computer can understand them. There are two ways this can be done: the code can be interpreted or compiled.

When code is interpreted the commands are converted into machine code and executed one at a time. Scripting languages such as Java Script, PHP, Python and others are typically interpreted. In a compiled language all the code is converted into machine code before the code is executed.

Compiled code is usually considered to be faster and more secured, but interpreted code is lighter and theoretically easier to debug because it always breaks on the line it is currently executing.

Some years ago Java introduced a variation of these patterns. Java code is converted into a byte code that resembles assembler language (a language very close to machine code). Then this byte code is compiled to the particular machine by a Java runtime. Traditionally, compiled code had to be compiled for use on a particular processer and operating system. Code written for Windows on Intel would not run on Linux with an ARM CPU. In fact it would not run on Windows with an ARM chip. The intermediate step of compiling to byte code allows a programmer to write the code once and compile to any machine by virtue of the runtime. It makes the program OS and CPU independent. Microsoft used this same process for their .Net programming languages including C#.