# 5. Programming concepts

## 5.1 Conventions and best practice

Meaningful identifiers, proper indentation and adequate comments all improve the readability of code and thus save money, time and effort in programming teams. Pragmatically as a student of this course, it is in your interests for your code to be quickly and easily understood by your teacher and exam markers.

### Code layout

Coding should follow tabbed indentations in blocks between braces.

### Comments

Commenting should be clear, consistent, and regular.

### Naming consistency

Names should be meaningful! When naming your identifiers, Java conventions are as follows

- Class name: Start with uppercase and be a noun. Animal, Genus etc.
- Method name: Start with uppercase letter eg. addToList, initialiseStack etc.
- Variable name: Start with lowercase letter eg. firstName, orderNumber etc.
- Package name: Lowercase letters data structures, GUIs etc.
- Constants name: Uppercase letter. RED, MAX_PRIORITY

### Variables

- Since variables describe an attribute or property, they are like generally adjectives or nouns.
- A noun based naming scheme generally works best.

### Functions

- A function should perform one job.
- Minimise side effects. This means the function should not alter data outside of it.
- Since a function performs an action, name it using a verb.
- Names should be descriptive and consistent.
- Outputs should be avoided, use return whenever possible

## 5.2 Pass by value / pass by reference

Functions or subroutines are a useful way of allowing a program to be modularised / divided into sub tasks. It is frequently the case that you will want your function to receive particular information that it acts on, and then have it return a result. The passing of information is generally referred to as passing arguments or parameters.

Programming languages have two different methods of passing parameters, and the method used can significantly affect how your program behaves. These two methods are pass by value and pass by reference.

```
function doubleIt( var number ) {
    number = number * 2;
    return( number );
}

function main() {
    var a = 10;
    b = doubleIt( a );
    print( a )
    print( b )
}
```

What will the above program print?

The answer is... it depends on if the programming language is using pass by reference or pass by value!

Pass by value will take the value stored within a, allocate new memory space for the number, and place a copy of the number 10 into that new memory space. Pass by reference will lookup the memory address used by a, and assign the variable number the same memory address.

Now... What will the above program print in each case?

Technically Java uses pass by value for everything. In practice, it will only seem like it behaves that way for primitive data types (integers, floats, booleans etc). For objects and complex data structures such as arrays, it will seem like it is using pass by reference. This is because internally within Java, the "value" of an array or object variable name is the memory address of where the array or object is stored.

The implication of this is that this will behave like pass by reference even though it is technically pass by value.

```
import java.util.Arrays;

public static int[] doubleIt(int[] data) {
    for (int i=0; i<data.length; i++) {
        data[i] = data[i] * 2;
    }
    return(data);
}
```

```java
public static void main(String[] args) {
    int[] numbers = { 1, 2, 3, 4, 5 };
    int[] numbers2 = doubleIt( numbers );
    System.out.println( Arrays.toString( numbers ));
    System.out.println( Arrays.toString( numbers2 ));
}
```

One solution to get around this problem in the above example would be to change the `doubleIt()` function as follows...

```java
public static int[] doubleIt(int[] data) {
    int[] result = new int[data.length];
    for (int i=0; i<data.length; i++) {
        result[i] = data[i] * 2;
    }
    return(result);
}
```

Why does this solve the problem?

It is important to be on guard for this issue and be aware of how it can impact your programming. When starting to get serious with any new programming language, knowing what is treated as pass by value, and what is treated as pass by reference is a question you will need to know the answer to.

## 5.3 Understanding code

How do computers understand code?
https://www.youtube.com/watch?v=QXjU9qTsYCc

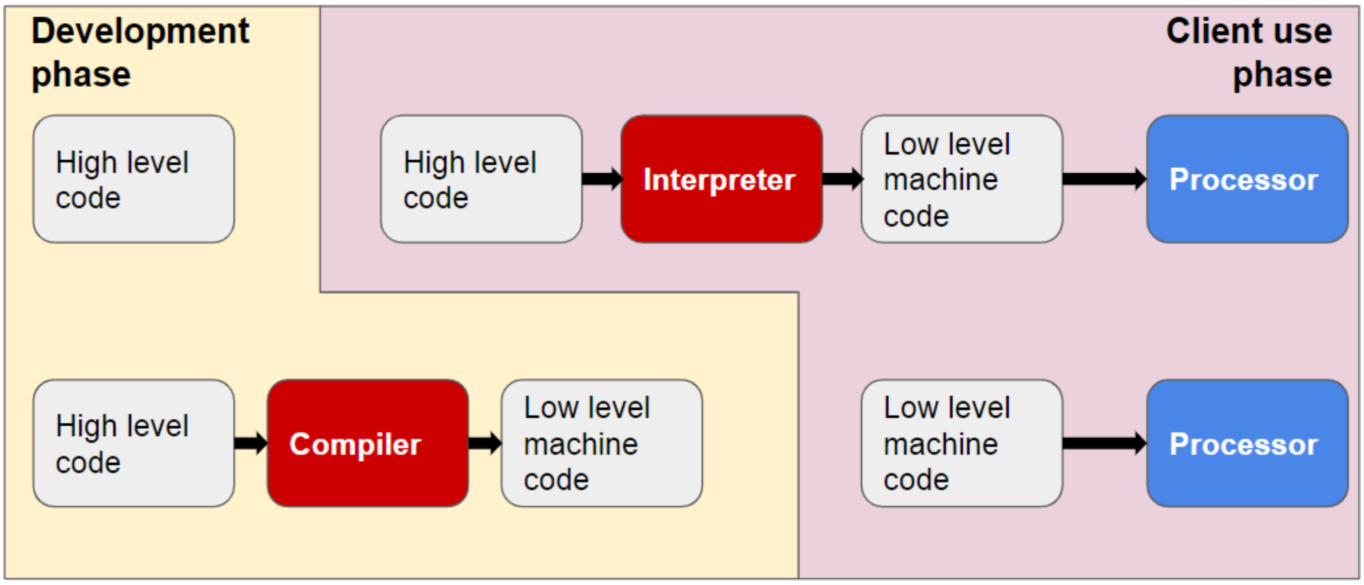
Software programs are sets of instructions. For a CPU to execute these instructions, each one must first be translated into machine code – simple binary codes that activate parts of the CPU.

The CPU only performs a few basic functions:

- performing mathematical operations like addition and subtraction
- moving data from one memory location to another
- making decisions and jumps to a new set of instructions based on those decisions

A piece of software, such as a game or web browser, combines these functions to perform more complex tasks. These are known as compound operations.

There are two main ways that our programs are converted into this machine code: interpretation and compilation.



**Compiler**

A compiler translates a human-readable program directly into an executable, machine-readable form before the program can run.

**Interpreter**

An interpreter translates a human-readable program into an executable, machine-readable form, instruction by instruction. It then executes each translated instruction before moving on to the next one. A program is translated every time it is run. Python is an example of an interpreted language.

## 5.4 High / low level languages

As we know, a computer program is a list of instructions that enable a computer to perform a specific task. The languages that we use to write our computer programs in are generally split into two categories, depending on the task and the hardware being used. These are high level languages and low level languages.

### High Level Languages

When we think about computer programmers, we are probably thinking about people who write in high-level languages.

High level languages are written in a form that is close to our human language, enabling the programmer to just focus on the problem being solved.

No particular knowledge of the hardware is needed as high level languages create programs that are portable and not tied to a particular computer or microchip.

These programmer friendly languages are called 'high level' as they are far removed from the machine code instructions understood by the computer.

Examples include: C++, Java, Pascal, Python, Visual Basic.

Advantages

- Easier to modify as it uses English like statements
- Easier/faster to write code as it uses English like statements
- Easier to debug during development due to English like statements
- Portable code – not designed to run on just one type of machine

Modern high level languages would come with features such as:

- Variables & constants
- Types - integers, floating point numbers, characters, strings, booleans
- Operators - add, subtract, multiply, divide, modulus, concatenate etc
- Loops - for, while, repeat until
- Branching - if, if else, else
- Collections - saving and retrieving multiple items to one variable identifier
- Arrays/lists - a subtype of collection
- Sub routines, functions
- Error or exception handling

### Low Level Languages

Low level languages are used to write programs that relate to the specific architecture and hardware of a particular type of computer. They are *closer* to the native language of a computer (binary), making them harder for programmers to understand. Low level languages include Assembly Language and Machine Code.

Assembly Language

- Few programmers write programs in low level assembly language, but it is still used for developing code for specialist hardware, such as device drivers.
- It is easily distinguishable from a high level language as it contains few recognisable human words but plenty of mnemonic code.
- Typical assembly language opcodes include: add, subtract, load, compare, branch, store

Advantages

- Can make use of special hardware or special machine-dependent instructions (e.g. on the specific chip)
- Translated program requires less memory
- Write code that can be executed faster
- Total control over the code
- Can work directly on memory locations

Machine Code

- Programmers rarely write in machine code (binary) as it is difficult to understand.