
PROGRAMMING
LANGUAGES

PROGRAMMING LANGUAGES

every instruction consists of two parts:
OPERATION CODE, specified the action to be carried out
OPERAND, memory address relevant to that instruction

PROGRAMMING LANGUAGE

a set of instructions written in a high-level language, which consists of short words generally representing instructions in English

the translation of instuctions expressed in human language into the language used and understood by computers, which is binary code

the language used as a means of communication between the user (the **programmer**) and the digital computer

PROGRAMMING LANGUAGES

The programmer plays a fundamental role because they are the instruments through which the requests of the users are understood and carried out by the computers. They need to know perfectly both the vocabulary, the set of instructions, the keywords and the vocabulary of the programming language chosen.

Their work consists of two steps:

STEP 1: the programmer writes the programme in the chosen programming language. The result is one or more text files containing the instructions. In this form, the programme is called **source code**.

STEP 2: the source code is subjected to a special programme called **compiler** which checks the correctness in terms of lexicon and syntax. If everything is correct, the compiler translates the source code into the language of the computer producing the **executable code**.
Each compiler works with its proper CPU.

PROGRAMMING LANGUAGES

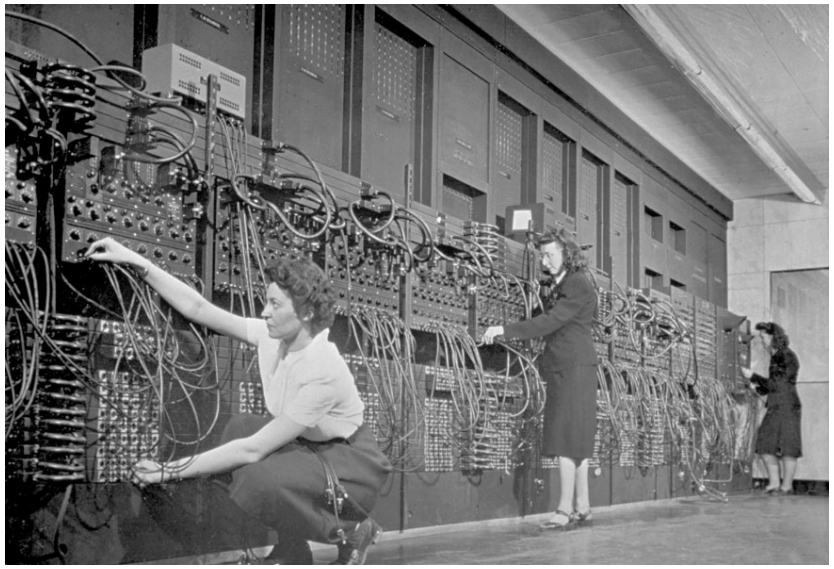
At the beginning of computer programming, in the 1940s, computers consisted of special-purpose computing hardware: each computer was designed to perform one specific arithmetic task or set of tasks.

First programmers had to manipulate parts of the computer directly, or they had to physically put switches in on or off positions.

The hardest part, though, was represented by the language used: programmers too used the binary code to write instructions. They used the machine language and not the human one.

The task written could only be performed by that computer only.

FIRST GENERATION MACHINE CODE



Instruction Address	*Opcode*	*Operand*	*Explanation*
00000001	10000000	00001110	Load contents of memory location num1 into accumulator.
00000010	01100000	00001101	Add the contents of memory location num2 to accumulator.
00000011	10100000	00001111	Store the result in memory location sum.

PROGRAMMING LANGUAGES

In the 1950s, programming made a step forward with **assembly language**: despite being still difficult to use, it is closer to human language than to machine language.

It consists of using convenient alphabetic abbreviations of English words called mnemonics to represent operation codes and abstract symbols to represent operands.

The computer cannot directly understand this language, so it needs translating in binary code before being executed.

Again, the task written could only be performed by that computer only.

SECOND GENERATION ASSEMBLY LANGUAGE

Mnemonic Opcode	*Symbolic Operand*
load	num1
add	num2
store	sum

PROGRAMMING LANGUAGES

Later developments of programming languages have been thought with the programmer in mind rather than the computer. These are **high level languages** and they aren't machine dependent, meaning that once a programme has been written it can be used on different computers. These programming languages are **people-oriented**.

The instructions in these languages are called **statements**, which resemble English phrases combined with the mathematical terms needed to express the task or problem programmed.

There are then **procedures-oriented languages**: in this case, programmers can write instructions in batch, grouping more instructions into one. A procedural language is one that expresses a computer problem as a series of discrete tasks and the instructions needed to accomplish those tasks. These languages are classified as "business, scientic or multipurpose".

THIRD GENERATION PEOPLE-ORIENTED PROGRAMMES AND PROCEDURES-ORIENTED PROGRAMMES

```
program sums2 (input, output);
var
num1, num2 sum : integer;
begin
        read (num1, num2) ;
        sum := num1 + num2 ;
        write1n (sum)
end .
```

High level languages, such as this sample of Pascal, brought computer programming closer to human thought processes.

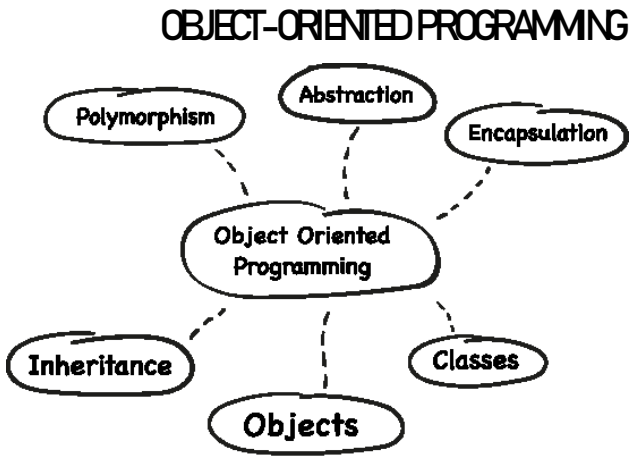
PROGRAMMING LANGUAGES

Object-oriented programming (**OOP**) is the last generation programming languages: it requires to code in terms of objects and their relationship to one another.
The fundamental concept of OOP is that there are **objects** (derived classes or subclasses) grouped in **classes** (superclasses or parent classes).

Every class contains more objects and they are somehow related: the process of creating these structures is called **data modelling**.

Every class possesses some **characteristics** which are passed down to its objects, which inherit the characteristics of the class and add new ones to them. This concept is called **inheritance**.

Two important features of OOP are polymorphism and encapsulation. **Polymorphism** can be defined as the "many forms of a single entity", which means that a single task can be performed in more ways (OOP can select the correct function depending on the context.



Encapsulation is the unity of data and methods to apply on data, which are shared both by classes and by objects. Data can only be accessed through the object which was designed to operate on that particular data (data are hidden from manipulation). This feature is also called **information hiding**.

PROGRAMMING LANGUAGES

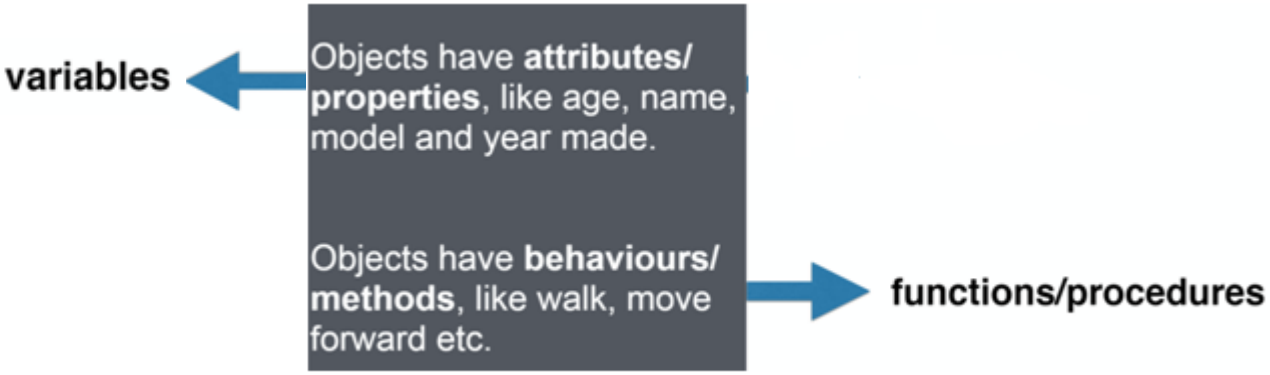
Objects are software bundles, which are often used to model the real-world objects, like people or car or robot etc.

Objects have **attributes/properties**, like age, name, model and year made.

Objects have **behaviours/methods**, like walk, move forward

A ***class*** is a ***blueprint*** or ***prototype*** from which objects are created.

A ***method*** is a subprogram associated with a class that defines certain behaviour of an object (i.e., a button object will change colour when clicked).



PROGRAMMING LANGUAGES

INHERITANCE

schoolMember

- student
  - **name**
  - **DoB**
  - form
  - tutor
- teacher
  - **name**
  - **DoB**
  - salary
  - department

subclass

schoolMember is the **parent class**
teacher and **student** will be **subclassing** the schoolMember class

Inheritance

Child class has access to the methods and attributes of the parent class