1. "Numbering systems and units of measurement"

Positional numbering systems

A numbering system is a way to express and represent numbers through a set of symbols. Numbers, since ancient times, have been a necessary tool for quantifying a set of elements.

Numbering systems can be classified based on two main criteria:

- **Base**: the base of a numbering system is the number of symbols used to represent numbers. The most common numbering system, the decimal one, has a base of 10, therefore it uses 10 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- **Placement**: a numbering system is positional if the value of a symbol depends on its position within the number. The decimal system is positional: for example, the number 123 means (1 x 100) + (2 x 10) + (3 x 1), or 123 units.

Modern numbering systems are positional, meaning all symbols (or digits) are arranged so that each has greater weight than the preceding symbol (digit): the value of the represented number depends on the relative positions of the digits that compose it.

Mostra immagine{width="13cm" height="11.194cm"}

There are also non-positional numbering systems, such as the Roman numeral system, which uses different symbols to represent different values. For example, the number 5 is represented by the symbol "V", the number 10 is represented by the symbol "X", the number 100 is represented by the symbol "C", the number 500 by the symbol "D", and the number 1000 by the symbol "M".

Binary numbering system

The binary numbering system is based on only two symbols: 0 and 1, and is therefore a positional numbering system in base 2.

When using different numbering systems, to avoid ambiguity, the base is written as a subscript of the number, so in this case, to indicate a binary number, 2 is put as a subscript, while to indicate a decimal number, 10 is put as a subscript.

As in all numbering systems, each digit has a weight that is determined by its position.

In computer science, the binary system is used for the internal representation of information by almost all electronic processors, as the physical characteristics of digital circuits make it very convenient to manage only two values, physically represented by two different levels of electrical voltage. These values conventionally assume the numerical meaning of 0 and 1 (like the current that passes or does not pass through a circuit) or those of true and false in Boolean logic.

Below is the corresponding conversion to binary number for the first twenty digits of the decimal system and the powers of 2 up to the tenth.

Conversion from decimal to binary

To convert a decimal number to binary, divide the decimal number by 2, writing only the integer part as a result and noting the remainder of the division. The procedure is repeated until 0 is obtained as the final result.

At the end, consider the remainders of the various divisions, reading them in reverse.

Example:

Mostra immagine{width="17cm" height="4.163cm"}

Calculating the remainder of a division

To understand how to calculate the remainder of a division, consider the following example:

Mostra immagine{width="6.6cm" height="4.5cm"}

The procedure for determining the remainder of a division is as follows:

1. Calculate the result of the division, writing only the integer part
2. Remainder = Divisor - (Dividend X Result)

Applied to the example, it becomes:

1. 750 : 16 = 46
2. 750 - (16 X 46) = 750 - 736 = 14

Therefore, the remainder is equal to 14

Conversion from binary to decimal

Since the binary system is "base 2", each digit of a number corresponds to a power of 2. To convert a binary number to decimal, multiply, starting from right to left, each composing digit by the power of two relative to the position assumed. In particular, the rightmost digit is in position 0, and as you move to the left, its position is incremented by 1.

The rightmost digit is called the least significant digit, while the leftmost digit is called the most significant digit. At the end, add the results of all the multiplications of the individual digits by the relative power.

Example:

Binary representation ** 111001~2~ ** ** **(1 · 2^5^)** + (1 · 2**^**4^)** **+ (1 · 2^3^)** **+ (0 · 2^2^)** **+ (0 · 2^1^)** **+ (1 · 2^0^)** **= **
**(1 · 32) + (1 · 16) + (1 · 8) + (0 · 4) + (0 · 2) + (1 · 1) = **
32 + 16 + 8 + 1
87
111001~2

Fixed-point representation

The representation of a value in fixed point is not much different from the representation of integers: we are still working with a positional numbering system, particularly with the binary one.

In this case too, each digit of a number will assume a value given by the value of the digit itself, appropriately weighted by the position in which the digit is located.

Negative numbers in binary

To represent negative numbers in binary format, a method called 'two's complement' is used.

Given numbers of N binary digits, if we want to represent both negative and positive numbers, we divide the range into 2 parts: negative, zero, positive, so we will have half of the numbers normally available.

With 8 bits, in classic representation (only positive numbers), we can represent 255 positive numbers plus zero, so from 0 to 255.

With 8 bits, in two's complement representation, we can represent 128 negative numbers, zero, and 127 positive numbers, thus filling the range of 256 numbers.

So the range will be -128 to 127.

In two's complement representation, the first bit expresses the sign:

- **0: positive sign**, representation identical to the classic one
- **1: negative sign**, two's complement representation

For example, the positive number **10**, in binary **00001010**, has a positive sign, in fact, the **first bit is 0**.

Instead, the number **-10**, in binary with two's complement is 1110110, and is obtained starting from the absolute value, 10, represented in binary:

00001010

to which then all bits are inverted (or 'complemented'):

11110101

to which then 1 is added:

11110101 +

00000001 =

_____

11110110

As we see, the number -10, negative, has the first bit as 1, which expresses the negative sign.

Byte

The byte is the unit of measurement of memory capacity, or in other terms, the measure of the amount of information.

To understand what a byte is, one must refer to the logic that governs the operation of computers. Computers operate using base 2 (binary system), which uses only two digits: 0 and 1. Each of these digits is called a bit.

The byte is composed of 8 bits and is therefore capable of assuming 2^8^ = 256 possible values (from 0 to 255). By definition, 1 byte, indicated by the capital letter B, is a unit of measurement of the amount of information and corresponds to a sequence formed by 8 consecutive bits.

The multiples of the byte are indicated as follows:

NAME SYMBOL MULTIPLE yottabyte YB 2^80^ zettabyte ZB 2^70^ exabyte EB 2^60^ petabyte PB 2^50^ terabyte TB 2^40^ gigabyte GB 2^30^ megabyte MB 2^20^ kilobyte kB 2^10^ **byte B** 2^0^ *bit bit* * 1/8*

Note that the byte, being formed by 8 bits and therefore by binary quantities, to go up or down a step uses the power of 2 and not that of 10 as can be for the meter or the kilo.

It follows that:

1 kB = 2^10^ B = 1,024 B

1 MB = 2^20^ B = 1 x 1,024 x 1,024 = 1,048,576 B = 1,024 kB

1 GB = 2^30^ B = 1 x 1,024 x 1,024 x 1,024 = 1,073,741,824 B = 1,048,576 kB = 1,024 MB

1 TB = 2^40^ B = 1 x 1,024 x 1,024 x 1,024 x 1,024 = 1,099,511,627,776 B = 1,073,741,824 kB = 1,048,576 MB = 1,024 GB

To perform a conversion between multiples of the byte, draw a scale with a series of steps, on which the symbols of the binary multiples of the byte are reported in order.

Mostra immagine{width="7.609cm" height="7.59cm"}

To perform any conversion, count the number of steps that separate the two units of measurement involved, and then:

- multiply by 2 raised to the number of steps by 10 if you go down;
- divide by 2 raised to the number of steps by 10 if you go up.

Example:

Convert 0.5 TB to MB

- You need to go down 2 steps, therefore:
- 0.5 x 2^2x10^ = 0.5 x 2^20^ = 0.5 x 1024 x 1024 = 524,288 MB

Convert 3,298,534,883,328 kB to TB

- You need to go up 3 steps, therefore:

- 3,298,534,883,328 : 2^3x10^ = 3,298,534,883,328 : 2^30^ = 3,298,534,883,328 : 1024 : 1024 = 3,072 TB

Bit

The bit is a binary digit and can contain the two symbols of the binary system: zero (0) and one (1) and is considered the smallest unit of information. Since the beginning of computing, the binary system has been the mother language of electronic processors and computers. The reasons for this choice are very simple. The "zero" and "one" states can be realized with the presence or absence of electrical voltage in a circuit.

To indicate a bit, the lowercase letter "b" or simply the word "bit" is used.

As mentioned earlier, 8 bits is equivalent to 1 Byte.

The multiples of the bit follow the same logic as the byte, and are indicated as follows:

NAME SYMBOL MULTIPLE yottabit Ybit 2^80^ zettabit Zbit 2^70^ exabit Ebit 2^60^ petabit Pbit 2^50^ terabit Tbit 2^40^ gigabit Gbit 2^30^ megabit Mbit 2^20^ kilobit kbit 2^10^ **bit bit** 2^0^

Note that the bit, being a binary digit, to go up or down a step uses the power of 2 (as for the byte) and not that of 10 as can be for the meter or the kilo.

It follows that:

1 kbit = 2^10^ bit = 1,024 bit

1 Mbit = 2^20^ bit = 1 x 1,024 x 1,024 = 1,048,576 bit = 1,024 kbit

1 Gbit = 2^30^ bit = 1 x 1,024 x 1,024 x 1,024 = 1,073,741,824 bit = 1,048,576 kbit = 1,024 Mbit

1 Tbit = 2^40^ bit = 1 x 1,024 x 1,024 x 1,024 x 1,024 = 1,099,511,627,776 bit = 1,073,741,824 kbit = 1,048,576 Mbit = 1,024 Gbit

To perform a conversion between multiples of the byte, draw a scale with a series of steps, on which the symbols of the binary multiples of the byte are reported in order.

Mostra immagine{width="7.609cm" height="7.59cm"}

To perform any conversion, count the number of steps that separate the two units of measurement involved, and then:

- multiply by 2 raised to the number of steps by 10 if you go down;
- divide by 2 raised to the number of steps by 10 if you go up.

Example:

Convert 0.5 Tbit to Mbit

- You need to go down 2 steps, therefore:
- 0.5 x 2^2x10^ = 0.5 x 2^20^ = 0.5 x 1024 x 1024 = 524,288 Mbit

Convert 3,298,534,883,328 kbit to Tbit

- You need to go up 3 steps, therefore:
- 3,298,534,883,328 : 2^3x10^ = 3,298,534,883,328 : 2^30^ = 3,298,534,883,328 : 1024 : 1024 = 3,072 Tbit

Binary information representation

For programming language purposes, it is common to group bit sequences into larger entities that can assume values in much wider ranges than that allowed by a single bit. These groupings generally contain a number of binary strings equal to a binary power, that is 2^n^; the most well-known is the byte corresponding to 8 bits, which constitutes the most used unit of measurement in the computer field. In other areas outside of programming, these groupings are little used.

Other groupings of this type are the following:

+------------+--------------------+ | **NUMBER** | **GROUPING TERM** | | **OF BITS**| | +------------ +--------------------+ | 1 | bit | +------------+--------------------+ | 4 | nibble | +------------ +--------------------+ | 8 | Byte | +------------+--------------------+ | 16 | word | +------------ +--------------------+ | 32 | double word | +------------+--------------------+ | 64 | quad word | +------------+--------------------+

Hertz

The hertz (symbol Hz) is the unit of measurement of the International System of frequency.

Frequency represents the number of cycles or oscillations completed in one second. So, one Hertz is equivalent to one cycle per second. For example, if something has a frequency of 10 Hertz, it means that it completes 10 cycles or oscillations in one second.

In computing, hertz have multiple uses, for example, they are used to measure the clock frequency of a processor or the refresh rate of a screen. For example:

- **The CPU frequency**, represents how many elementary operations (or instructions) a processor can execute within one second.

A CPU with a higher frequency can execute more instructions per second, which means it can process information faster.

A CPU with a clock frequency of 3 GHz can execute 3 billion elementary operations per second.

- **The refresh rate of a monitor** is the number of times per second the image is redrawn (or updated) on the screen. A higher refresh rate produces a smoother image.

A screen with a refresh rate of 144 Hz redraws the image 144 times in the course of a second.

Use of the binary system

The binary system is used for the internal representation of information by almost all electronic processors, as the physical characteristics of digital circuits make it very convenient to manage only two values, physically represented by two different levels of electrical voltage. In addition to the internal use made by a computer, the study of the ASCII code is of particular interest, which allows to understand the amount of information that can be represented with 1 byte (that is, 8 bits which are equivalent to 256 combinations).

ASCII code

To allow communication between humans and computers without having to resort to binary code, the ASCII coding system was introduced. This system associates each of the possible 256 values assumed by the byte with a specific letter, number, or special character. For example, the uppercase letter A corresponds to the byte 01000001.

The acronym "ASCII" stands for American Standard Code for Information Interchange.

To perform this coding (or conversion), the ASCII table is used, which simply associates each character with a numeric code.

The table consists of 256 decimal numbers ranging from 0 to 255 (from 00000000 to 11111111 in binary).

- Numbers from 0 to 31 are reserved for control signals.
- Numbers from 32 to 127 constitute the standard character set, i.e., all the letters of the Latin alphabet in lowercase and uppercase, numbers, and the most used symbols.
- Numbers ranging from 128 to 255 constitute the extended character set which includes special, mathematical, graphic, and foreign language characters (such as æ and Ç).

The ASCII table is organized in three columns:

- the first column contains the corresponding binary number associated with a particular character;
- the second column contains the corresponding decimal number associated with a particular character;
- the third column contains the corresponding character.

For example, the letter "A" has ASCII code 65, the number "1" has ASCII code 49, and the symbol "@" has ASCII code 64.

Standard ASCII Table Mostra immagine{width="16.579cm" height="15.476cm"}

Combinations from 0 to 31 may be different depending on the software used

If you are using Windows, you can get any ASCII character by holding down the Alt key and typing the corresponding decimal code with the numeric keypad (if the numeric keypad is not active, press the "Num lock" key to activate it).

for example, the @ symbol is obtained by typing 64 while holding down the Alt key.

It is useful because, for example, in the keyboard with Italian layout, some symbols are missing, such as the apostrophe (96), curly brackets (123,125) or tilde (126) and accented capital letters (À, È, Ì, Ò, Ù).

Extended ASCII Table Mostra immagine{width="16.805cm" height="18.06cm"}

ASCII code expansions

The first version of the ASCII code was 7-bit, meaning it provided 128 combinations (shown in the first part of the table.

The ASCII code has been extended to support additional characters for international language encoding. The most common extensions are:

- **Extended ASCII**: adds 128 additional characters, for a total of 256 characters.
- **Unicode:** adds 1,114,112 characters, for a total of 1,114,112 characters.

Size of a text file with ASCII encoding

If, for example, you create two files with a text editor (with a program like Windows Notepad) and write the words "HELLO" and "JOHN" inside them, both files would occupy 5 bytes and 4 bytes respectively, since each character occupies exactly 1 byte and these words written inside the file are composed of 5 and 4 characters respectively. This is because 1 byte is formed by 8 bits and therefore by 256 combinations that represent symbols/characters.

Mostra immagine{width="8.31cm" height="9.88cm"}Mostra immagine{width="8.31cm" height="9.88cm"}

The information on the composition of the characters is contained in the bit sequence of each single byte, in fact, we can refer for each character to the corresponding number in binary contained in the ASCII code.

Taking the example just made, HELLO corresponds to:

+----------+----------+----------+----------+----------+ | 72 | 69 | 76 | 76 | 79 | +----------+---------- +----------+----------+----------+----------+ | that is | 01000101 | 01001100 | 01001100 | 01001111 | | | | | | | | 01001000 | | | | | +----------+----------+----------+----------+----------+

While JOHN corresponds to:

+----------+----------+----------+----------+ | 74 | 79 | 72 | 78 | +----------+----------+----------+----------+ | that is | 01001111 | 01001000 | 01001110 | | | | | | 01001010 | | | | +----------+----------+---------- +----------+

Use of the hexadecimal system

The hexadecimal system is widely used in computing, due to its direct relationship between one hexadecimal digit and four binary digits.

It is used in different programming languages, especially to identify memory addressing, a particular use occurs in graphics programs to identify a specific color with few hexadecimal digits.

HTML code

As mentioned earlier, color codes are hexadecimal triplets (and therefore a digit composed of 6 hexadecimal digits), which represent the colors red, green, and blue (the three primary colors).

The hexadecimal base is used by most graphics programs and by various browsers for internet navigation, to unequivocally identify a very specific color.

The hexadecimal number with the highest value is FF which is equivalent to the decimal value 255, while the lowest hexadecimal number is equivalent to 00.

Colors are indicated with a 6-character code preceded by the # symbol that indicates a color. This code composed of 6 hexadecimal digits is also called "Hexadecimal color code".

Let's examine the white color, in hexadecimal base it is indicated as `#FFFFFF` and is composed of:

- FF, i.e., Red = 255
- FF, i.e., Green = 255
- FF, i.e., Blue = 255

255 parts of red, 255 parts of green, and 255 parts of blue

Below are some examples of colors with the corresponding hexadecimal code.

NAME CODE COLOR White `#FFFFFF`
Black `#000000`
Red `#FF0000`
Blue `#0000FF`
Green `#009900`
Yellow `#FFFF00`
Magenta `#FF00FF`
Purple `#5C2E91`
Azure `#0080FF`

---

1. "Hardware"

Computer hardware

The computer is a set of connected physical devices that have the purpose of receiving data from the outside, processing it according to certain instructions contained in programs, and producing results at the end of the processing process.

Like all machines, it has no decision-making or discretionary capacity, but it simply performs certain actions according to pre-established procedures (programs).

To simplify as much as possible, we can define a computer as composed of the following components:

1. Hardware Components (the physical and tangible parts of the computer)
2. Software Components (the logical parts of the computer such as programs and applications)

It follows that, in a computer, the hardware and software components are complementary to each other and indispensable for it to work.

Mostra immagine{width="6.63cm" height="6.63cm"} Mostra immagine{width="9.299cm" height="5.761cm"}

\
On the left an example of hardware components, on the right an example of some software.

Hardware refers to all the physical part of a computer, that is, all those electronic, electrical, mechanical, magnetic, optical parts that allow its operation.

The various hardware components of a computer are formed by electronic circuits, capable of processing the two states of electricity crossing corresponding to the symbols 0 and 1.

To be considered as such, a computer must have at least:

- CPU;
- Memory;
- Peripherals.

Von Neumann Architecture

All modern computers are based on the architecture of the von Neumann model:

Mostra immagine{width="13.494cm" height="7.805cm"}

The scheme is based on five fundamental components:

- **CPU** (or processor);
- **Memory** (where data is temporarily or permanently stored);
- **Input unit**, through which data is entered by the user into the computer to be processed;
- **Output unit**, necessary so that the processed data can be returned to the user;
- **Bus**, a channel that connects all the components together.

His model, conceived in 1945, still constitutes the foundation for almost all modern computers today.

CPU

The CPU (Central Processing Unit) has as its fundamental task the execution of the instructions of the programs that the computer runs.

It uses a series of components, internally, which perform specific tasks:

- ALU: Arithmetic Logic Unit
- CU: Control Unit, coordinates the acquisition and execution of instructions
- Clock: a system clock that synchronizes components and instructions.
- Internal registers: temporarily store instructions and partial results of operations.

To be executed, a program must be loaded into RAM memory from the mass memory where it is stored permanently.

Once the program resides in RAM memory, an instruction at a time is cyclically executed, the single instruction. to be executed, it must be taken (**Fetch**) from RAM memory to the CPU, where it is encoded (**Decode**) and then executed (**Execute**).

The **Fetch-Decode-Execute** cycle is called the processor cycle (or clock) and is executed cyclically by the CU, while the ALU performs calculation operations when necessary.

Mostra immagine{width="7.8cm" height="6.35cm"}

Among the characteristics of a CPU to remember:

- Clock frequency: is the speed of the CPU, measured in multiples of Hertz (Hz), currently between 2 and 5 GHz
- Number of cores: single, dual, quad, octa
- Number of threads
- Presence and type of cache: L1 (inside the CPU, fast), L2 (inside or outside CPU), L3 (between CPU and RAM)
- Architecture (or instruction set): x86, x86-64, ARM, PowerPC, MIPS, SPARC, etc..

Clock frequency

The clock frequency (or clock speed) represents the number of elementary operations processed in the course of a second and is measured in GHz (gigahertz).

In general, a higher clock frequency means the CPU is faster. The CPU processes many instructions every second (low-level calculations, such as arithmetic) from different programs.

For example, a CPU with a clock speed of 3.2 GHz executes 3.2 billion cycles per second (or more simply 3.2 elementary operations in one second). In the past, CPU speed was measured in megahertz and therefore in millions of cycles per second.

Number of cores

A multi-core CPU is a set of autonomous processing units in a single chip, which share the various system resources (cache, bus, and central memory).

In theory, having, for example, two cores means having a duplicated processing power, but only if the applications are optimized to be processed by a multi-core CPU.

In other words, a multi-core CPU has the ability to execute multiple instructions simultaneously in parallel, in theory, a dual-core CPU at 2 GHz has the ability to execute 4 billion elementary operations per second

Also, a multi-core CPU does not need more energy than a single core, resulting in greater computing power and also greater autonomy for mobile devices.

Number of threads

A multi-thread CPU represents the ability of each single core to simultaneously process multiple elementary operations.

A thread, to greatly simplify, is a part of the instructions that a CPU must process to perform a task (or, better, a "process"). A multi-thread CPU can therefore subdivide a process into multiple threads and process them in parallel to provide the expected result more quickly.

Even if the CPU is only one, the operating system sees two "logical units" because it can actually process two threads simultaneously. In multi-core CPUs, the number of threads refers to each core of the processor. For example, a quad core/dual thread CPU is capable of processing 8 operations simultaneously as it has 4 physical cores (and therefore 4 simultaneous operations) and each single core is seen as 2 separate cores, it follows that 4 cores x 2 threads = 8 simultaneous operations.

CPU cache

The cache used by a computer's CPU has the task of reducing the average memory access time; it is a type of memory of reduced size, but very fast, which keeps copies of the data that is most frequently accessed in the more capacious main memory.

On a CPU there can be 3 cache levels and they are organized hierarchically:

The L1 cache (first level cache): it is internal to the processor and is often differentiated into data cache and instruction cache. The L1 cache tends to be on the order of 32-64 kB.

The L2 cache (second level cache): it is generally larger but slightly slower than the L1 cache and is linked to a core of the CPU. Recent processors tend to have up to 512 kB of cache per core and this cache does not distinguish between instructions and data cache as it is a unified cache.

The L3 cache (third level cache): it is shared by all the cores present on the CPU and is much larger and slower than the L2 cache, but is still much faster than the main memory. It can be either internal or external but may also not be present (especially in less recent CPUs). It is not differentiated between data and instructions. The L3 cache tends to be on the order of 16-32 MB.

Architecture (Instruction set architecture)

The architecture of a CPU (in English, instruction set architecture, the acronym of which is ISA) defines the set of basic instructions (or elementary operations) that the CPU can perform and which therefore constitute its machine language, from which the related programs are written in the various programming languages.

The architecture of a CPU can be classified in different ways. A common classification is by architectural complexity. A computer with a complex instruction set (CISC) has many specialized instructions, some of which may be used only rarely in practical programs. A computer with a reduced instruction set (RISC) simplifies the processor by efficiently implementing only the instructions frequently used in programs, while less common operations are implemented as a combination of more elementary operations, with the consequent additional processor execution time compensated by infrequent use and with the advantage of drastically reducing energy consumption (a very useful factor for mobile devices such as smartphones and tablets)

Currently all computers (with the exception of Macs produced from 2020 onwards) mount processors with an instruction set of the "**x86**" type (families of products made by Intel and AMD, containing CISC-type instructions), while all smartphones and Macs produced from 2020 onwards use processors with an instruction set of the "**ARM**" type (families of products made by third-party manufacturers under license from ARM Holdings, containing RISC-type instructions).

In addition to these two architectures, there are others, but they are now obsolete in the consumer field, such as the PowerPC architecture, used for Macs until 2006 or by PlayStation 3 or Xbox 360.

Memory

Memories are intended for the saving of data and reading them. In computer architecture, two types of memory are distinguished:

- **main memory (or central memory)**, intended to work in close contact with the CPU, consisting fundamentally of RAM memory, ROM memory, Cache memory;
- **secondary memory (or mass memory)**, intended to store data permanently, such as hard disks, SSDs, but also removable media such as CDs, DVDs, flash memory of all types, USB sticks and more.

They are characterized by the following parameters:

- **Capacity**: amount of available space
- **Volatility**: indicates whether the memory can or cannot maintain the data in the absence of power
- **Access time**: time interval necessary to complete a read or write
- **Transfer speed**: amount of data transferred in a unit of time
- **Cost per bit: **price

The amount of memory space is measured in bytes and multiples.

Main memories

RAM (Random Access Memory)

RAM (Random Access Memory) temporarily stores data waiting to be processed by the CPU. In fact, it contains, during the execution of programs, the instructions of the programs that, gradually, the CPU must execute.

It is a **volatile memory**, that is, the data is lost when the system is turned off or restarted, and it is called **random access, **since the access time is independent of the physical position of the data.

The maximum limit of installable RAM is given by the processor architecture and the operating system used:

- 32 bit = 2^32^ addresses = 4 GB
- 64 bit = 2^64^ addresses = 16 EB (Exabytes or 16,000,000 GB)

Mostra immagine{width="13.651cm" height="3.401cm"}

ROM (Read Only Memory)

ROM is a read-only memory, it is therefore not possible to store data in it.

ROM contains all the configuration parameters, in order to make all hardware and software components interact with each other.

Inside the ROM, particular programs are stored:

- **BIOS (Basic I/O System)**, present on all computers produced until 2010, which is executed by the CPU when the computer is turned on, when the RAM is empty;
- **UEFI (Unified Extensible Firmware Interface)**, which is practically an evolution of the BIOS, as besides correctly starting the device and making the hardware dialog with the software, it integrates other advanced functions;
- **Firmware**, present on all mobile devices (such as smartphones and tablets) and on all embedded systems (such as household appliances in general, for example washing machines, dishwashers, microwave ovens, televisions, decoders, etc...)

The purpose of the BIOS (like any other firmware) is to make the various hardware components interact through the implementation of communication protocols or programming interfaces. It represents in fact the meeting point between the logical and physical components of an electronic device, i.e., between software and hardware.

In personal computers, the BIOS starting from 2010 has been progressively replaced by UEFI (Unified Extensible Firmware Interface), which is practically its evolution, as besides making the device correctly boot and making the hardware dialog with the software, it integrates other functions:

- **Operating system license management: **function of registering the digital license of the operating system in a specific storage area of the chip, so as not to have to reactivate the operating system following, for example, a formatting.
- **Boot manager:** or boot manager, which was previously stored only on the mass memory.
- **Secure boot:** or secure boot, which allows to block and prevent viruses or other threats from being started at the machine's power-up/restart, constituting a serious risk for the system.
- **Manufacturer warranty:** The platform allows the machine manufacturer to store the terms of any warranty provided, primarily the expiration date. In this way, the user can monitor the warranty status both through the firmware menu and through the manufacturer's tools (application or website).

Secondary memories

Hard Disk

A hard disk or fixed disk, (in English hard disk drive, commonly abbreviated as hard disk and with the acronyms HDD, HD) indicates a magnetic type of mass memory device that uses one or more magnetized disks for the storage of data and applications (files, programs and operating systems).

Like all mass memories, its primary purpose is the permanent storage of information inside it. It has a higher capacity than RAM but is considerably slower in terms of transfer speed.

Main features:

- Speed: of the hard disk "platters", 5400, 7200, 10000, 15000 rpm (revolutions per minute);

- Size: 2.5" (laptops) or 3.5" (desktop);
- Capacity: 500GB, 1TB, 2TB, 4TB, etc;
- Connection interface: SATA if internal, USB or Thunderbolt if external

Secondary features:

- Access time: average time needed to retrieve data (e.g.: with 7200 rpm HD, about 9 ms);
- Transfer speed: expressed in MB/s, indicates the amount of data provided by the HD in one second;
- Noise emitted, expressed in dB;

Mostra immagine{width="12cm" height="6.001cm"}

SSD (Solid State Drive)

A solid-state drive (abbreviated as SSD from the corresponding English term solid-state drive) is a semiconductor-based mass memory device that uses solid-state memory (solid-state storage) such as flash memories for data storage.

It performs the same function as the traditional hard disk, with some advantages:

- No noise, as there is no mechanical component (motor and magnetic disk) of rotation, unlike traditional HDs;
- Less possibility of failure;
- Lower power consumption during read and write operations;
- Reduced access and storage times: we work in the order of tenths of a millisecond (SSD: 0.1 ms, HDD: 5-10 ms);
- Higher data transfer speed (600 MB/s with SATA 3, 3 GB/s with PCIe 3.0);
- Greater resistance to shocks, as there are no mechanical parts inside it;
- Less heat production;
- Connection interface: SATA or PCI-Express (PCIe), in particular SATA SSDs have the same identical shape, size and connection interface as 2.5" SATA hard drives and are therefore interchangeable with them without installing specific hardware or software components;

Mostra immagine{width="14.469cm" height="4.461cm"}

Optical drives

The optical disc is a type of memory support. It consists of a flat, thin disc usually made of transparent polycarbonate. Inside it is inserted a thin metal sheet, usually aluminum, on which information is recorded and read through a laser beam.

The information on an optical disc is stored sequentially in a continuous track in a spiral, from the innermost track to the outermost one. Optical discs are particularly resistant to atmospheric agents.

From 2005 onwards, their use has progressively declined, due to other more compact and faster devices such as portable hard drives or pen drives. There are different types of optical discs:

I/O peripherals

Peripherals are hardware devices of the computer that allow the user to interact with the computer. It is called peripheral because the device is generally an external component of the motherboard that contains the CPU and is connected via cable or via a wireless connection. The computer peripheral is an electronic device that is connected to the computer to receive or send data, respectively peripheral (or device) of input and output.

In the hardware architecture of the computer, peripherals manage the flow of data into (input) and out of (output) the computer. All peripherals are connected to the CPU, which has the task of controlling the flow of data to and from the peripherals.

The peripherals of a computer are classified into:

Input peripherals: Input peripherals allow the user to enter data or to request the execution of a command or a program, to select a function, etc. They are peripheral input units. The data flow moves from the user to the computer. Some examples of input peripherals are:

- Mouse
- Keyboard
- Joystick
- Scanner
- Microphone

Output peripherals: Output peripherals allow the user to observe the result of data processing. They are peripheral output units. The data flow moves from the computer to the user. Some examples of output peripherals are:

- Monitor
- Printer
- Speakers
- Earphones

Input/output peripherals: Input/output (I/O) peripherals can perform both data reading operations (input) and data writing operations (output). Some examples of I/O peripherals are:

- USB sticks
- Touchscreen screens
- Modem/Router

Bus

The bus is a communication channel that allows peripherals and internal components of a computer to interface with each other by exchanging information or data of various types through the transmission and reception of signals.** **

Motherboard

The hardware components of the computer are designed around a main board that provides some integrated circuits and many electronic components such as capacitors, resistors, etc. All these components are soldered on the board and are connected by the printed circuit connections and by a large number of connectors: this board is called motherboard.

On the motherboard are present the socket for the processor (also called socket), the slots to install RAM memory, the slots for the card and for any expansion cards, the connectors for USB ports and the connectors to connect mass memories such as hard disks, SSDs, or DVD units.

Moreover, they can include a certain number of multimedia and network peripherals that can be deactivated:

- Integrated video card;
- Integrated audio card;
- Integrated network card.

Mostra immagine{width="8.17cm" height="12cm"}

Case

The case indicates the container inside which the main components of a computer are mounted.

The motherboard is mounted inside a case that also provides internal accommodation for mass memory devices (such as hard disks or DVD units), externally there are buttons that allow to turn on or restart the computer and a certain number of lights (or LEDs) that allow to verify the operating status of the equipment and the activity of the hard disks. In the back, the case provides openings at the level of the extension cards in addition to the spaces dedicated to the motherboard itself for the interfaces (ports) dedicated to keyboard, mouse, printers, etc..

Mostra immagine{width="9.2cm" height="12cm"}

Finally, the case hosts an electrical power supply block (commonly called power supply), which must provide a stable and continuous electric current to all the elements that make up the computer. The power supply therefore serves to convert the alternating current of the electrical network (220 Volt) into 5 Volt continuous voltage for the computer components and 12 Volt for some internal peripherals (disks, CDROM readers, etc.). The power supply block is characterized by its power, which conditions the number of peripherals that the computer is capable of powering. The power of the power supply block is generally between 250 and 800 Watts.

1. "Software"

Definition of software

Software has the important functionality of acting as an intermediary between the user and the hardware.

In this way, the user interacts directly (at a logical-functional level) with the software and, consequently, indirectly (at a physical level) with the hardware. The user is thus independent of the knowledge of the complex mechanisms that characterize it.

In practice, software is the logical part of a processor and is formed by sequences of instructions that guide the hardware in the execution of its tasks, and that unequivocally define the execution of a certain task. Generally, software means the set of programs used in a data processing system that manages the operation of a processor.

It is divided into two main components:

- Basic software (operating systems);
- Application software (programs or apps)

Basic software

Basic software, also known as system software, is software capable of managing all the hardware resources of any computer. In simple words, it is the part of the software closest to the machine's hardware. Speaking summarily, basic software is divided into three different categories:

- operating systems, such as Windows, Linux, Android or macOS;
- compilers and interpreters;
- and libraries.

Application software

Application software, also known as application (or app), is the set of programs that help the user solve a wide range of problems. In other words, these are all those applications that are not part of the operating system but are still necessary for the user to enable one or more specific functionalities.

Almost all software belongs to this category (with the exception of operating systems), a summary subdivision can be made based on the use for which it is intended, without prejudice to the fact that each category can be subdivided into many other sub-categories and so on:

- **productivity software**: this category includes, for example, word processors, spreadsheets, presentation software, database management, graphics and photo retouching software, email clients, etc.;
- **entertainment software**: video games for computers or consoles, emulators, audio and video players;
- **educational software:** that is, all those applications useful for schools of any grade;

- **scientific software**: that is, all those applications useful in the various fields of applied science, such as astronomy, biology or chemistry;
- **development software (software development):** includes programs and applications that developers use to create, to detect bugs (debug) or to modify other software, such as the so-called IDEs;
- **management software**: all those applications designed mainly for companies, such as personnel management, warehouse, invoicing, decision-making processes, CRM, ERP, OLAP, project management and e-commerce, etc.;
- **utilities**: it is a type of software that can be used for the analysis, configuration, optimization, and/or maintenance of the computer. Its function is to perform the configuration and/or monitoring of hardware or software devices. Examples can be: antivirus, clipboard managers, diagnostic programs, package managers, etc.

To create software, the programmer must write the **source code** in a specific programming language (Java, C++, Python, etc.). The source code must be translated into machine language to be executed, that is, to be transformed into an executable program (this operation is carried out in turn by a program, called a **compiler**). Software is often sold in the form of an executable program, first of all to prevent it from being modified and resold.

Mostra immagine{width="14.18cm" height="8.451cm"}

In reality, software can also be classified according to various characteristics that can be varied, here are some examples:

- **license** (free software or proprietary software);
- **operating system on which they can be used** (Windows, Android, Linux, etc.);
- **to be installed** or **portable**;
- **stand alone** (i.e., that can run completely autonomous on isolated systems) or **network** (i.e., that work in a network environment).

Software licenses and distribution

A license, in computing, is the contract with which the holder of the economic exploitation rights on the software defines the legal regime of circulation and the limitations in the use.

When you install software, in most cases it is necessary to accept the license before you can use it.

It is fundamental to read the license carefully before using any app since its acceptance puts the user in compliance with copyright laws, in fact in a proprietary software the license allows the beneficiary to use it under particular conditions and preventing others such as the study, modification, sharing, redistribution or reverse engineering. The restrictions are imposed by the holder of the economic exploitation rights, (the author or the transferee of the rights in question), through primarily legal means, such as cost of use, installation on a limited number of devices, or the circulation of the source code.

In practice, when you buy a software, you do not become the owner but you purchase the authorization to use it (with any limitations).

Classification of licenses

Software can be classified according to various characteristics, in particular according to its license, it can be divided into 2 categories:

- proprietary software (closed source)
- free software (open source)

Proprietary software

Proprietary software is software protected by copyright laws and in some cases also by patent, distributed through the subscription of a license, generally for a fee.

Free software

Free software (also called open source), provides for free use thanks to a copyleft license. Open source software is made such by means of a license through which the rights holders encourage the modification, study, use and redistribution of the source code. The most common licenses are:

- **GNU GPL (GNU General Public License):** it is a copyleft license for free software, Basically it establishes that a work protected by GNU GPL must remain free, that is, with the succession of modifications it must continue to guarantee to its users the so-called four freedoms:
    1. Freedom to run the program for any purpose;
    2. Freedom to study how the program works and to modify it according to your needs;
    3. Freedom to redistribute copies of the program in order to help others;
    4. Freedom to improve the program and to distribute your improvements publicly, so that the whole community benefits from them.
- **CC (Creative Commons)** which allows the copying and distribution of software as long as it is not for profit.

Software is created by companies called software houses and is a work protected by Legislative Decree no. 518 of 29/12/1992. According to Law no. 633 of 22/04/1941 (on copyright) the author of a program has the exclusive right to perform or authorize:

- the permanent or temporary reproduction, total or partial, of the computer program by any means or in any form;
- the modification of the program, as well as the translation and adaptation;
- the distribution of the original program or copies thereof.

1. "Digital processing of images and sounds"

Introduction

Digital image processing is an activity that, through the use of specific programs and electronic devices such as computers and tablets, allows to modify a digital image by making visible and appreciable variations to the observer. It is a common activity in the professional photographic field.

Furthermore, it also refers to all operations performed on an image to transform it, in order to make the extraction of information regarding the objects contained in it easier, to reproduce it or transmit it.

Images can be classified according to the method by which they are generated, such as photographs, drawings, paintings, television images, etc.

From all types of images, the corresponding numerical or digital images can be obtained by generating f(x,y) functions that measure the light intensity at the points with coordinates x,y of the image itself.

A digital image is not directly visible. It is a concept that acquires its meaning when it is possible to visualize it with adequate equipment. In practice, the data that form a digital image must be returned, after processing, on a screen.

Mostra immagine{width="17.069cm" height="3.81cm"}

Analog and digital

When talking about computers, tablets, smartphones, we always talk about "digital" and, sometimes, we contrast this term with that of "analog".

"**Analog**" and "**digital**" are terms that we continuously encounter when talking about technologies (old and new). In the common sense, "analog" is associated with a meaning of "old" or "past" or "low quality"; while "digital" is, instead, synonymous with "new" or "innovative" or "quality". This distinction from common sense *IS NOT true*.

First of all, it should be kept in mind that when we talk about analog and digital, we refer to the ways of representing the measurement of a quantity, to ways in which the quantities that we want to take into consideration vary (such as an audio signal, an image, a video signal, color, ...).

Mostra immagine{width="17cm" height="5.173cm"}

By **analog** we mean quantities that assume values in a continuous set, in particular an analog variable can assume an infinite number of values (for example, the distance between two points in space can assume an infinite number of values).

By **digital** we mean quantities that assume values within a set of discrete dimensions, in particular a digital variable can assume only a finite number of values (the duration of a day -- for example, can assume only one of the 3,600 values if we use minutes, or one of the 86,400 values if we use the "second" unit, or one of the 864,000 values if we used tenths of a second or one of the 8,640,000 values if hundredths of a second are used; many possibilities but still finite, determined).

Types of digital images

There are two types of digital images: raster (or bitmap) images and vector images. The former are constituted by a "map" of bits, while the latter are the result of mathematical functions.

Raster (or bitmap) images

Raster images are** **also called **bitmap** images, in this type of images, the stored values indicate the characteristics of each point of the image to be represented (pixel). In color images, the intensity level of the fundamental colors is usually stored (in the RGB color model, one of the most used, there are three: red, green and blue. Another example is CMYK, used for printing, based on four fundamental colors: cyan, magenta, yellow and black) while in monochromatic images in grayscale (improperly called black and white) the value indicates the intensity of gray, which varies from black to white

Advantages of raster graphics

- Rich gradation of colors and brightness;
- You can modify each single point.

Disadvantages of raster graphics

- Loss of quality when the image is scaled (enlarged);
- Compression can lead to a loss of quality;
- A lot of space is required to store a quality image;
- Vectorization involves a great computational expenditure.

Vector images

A vector image is described as a set of geometric primitives (points, lines, segments, polygons, etc.) to which colors and even shades can be attributed.

These images present two great advantages: they can be resized without loss of quality and they occupy little memory.

In other words, it can be said that this type of images are obtained from the union of a certain number of points or nodes, which form lines and polygons, in turn united in more complex structures, until forming the desired image.

This type of image is used in technical drawing for architectural and industrial design, in the representation of characters in word processors, in graphics for the creation of logos and brands or other objects, etc...

Advantages of vector graphics

- Scalable without loss of quality;
- Compression without loss of quality;
- Reduced file sizes;
- Easy management of any changes;

- Conversion to raster image is simple;
- Vector graphics, being defined through mathematical equations, is independent of resolution, while raster graphics, if enlarged or displayed on a device with a resolution greater than that of the monitor, loses definition.

Disadvantages of vector graphics

- Not suitable for complex graphic representations;
- The creation of vector images is not an intuitive activity as in the case of raster images;
- Use of advanced tools to create complex vector images;
- Resources adequate to the complexity of the image: a very complex vector image can be very substantial and require the use of a very powerful computer to be processed.

Mostra immagine{width="14.379cm" height="7.74cm"}

Elements of a raster image

In a raster digital image, the basic elements are:

- **Pixel:** elementary unit of information displayed on a computer screen
- **Graphic resolution:** Indicates the linear density of an image
- **Screen resolution**: Indicates the size (width x height), and is expressed in Pixels
- **Color depth**: Indicates the number of bits used to indicate the color of a single pixel.
- **Aspect ratio**: Indicates the ratio between the width and height (always expressed in pixels) of the digital image.

In a raster image, operations such as:

- **Processing: **In which digital images can undergo a processing process aimed at manipulating/modifying the image itself according to the desired characteristics (eg photo retouching, image restoration, etc.).
- **Compression: **As a special case of digital image processing, digital images can in turn undergo, according to various possible standards, an image compression process aimed at reducing the amount of associated information for transmission on the communication channel or storage on storage media (in simple terms to occupy less space while maintaining or not a certain quality)

Pixel

All output devices reproduce images using pixels (abbreviation of "picture elements" or in Italian "image element"), which are elementary units of display on screen (in practice it represents the conventional minimum unit of the surface of a digital image). They are arranged to compose a fixed rectangular grid, which for their smallness and density appear fused into a single image.

Simplifying it all, it can be said that pixels are small squares that represent a single color and that, close to each other, make up the image, similar to a mosaic.

The color of each single pixel is defined as a combination of the three primary colors: blue, red, green. This is not the only way to define a color, there are other ways that are called color spaces, but in the case of computer-generated images, the RGB (RED Red, GREEN Green, BLUE Blue) system is the most widespread given that graphics cards use it natively to generate the signal to be displayed with the monitor.

Graphic resolution

Graphic resolution is the quantity that quantifies the degree of detail of an image, given by the number of *image points* that linearly compose it (height or width).

The resolution of a raster image depends on the density of the pixels that compose it. The concept of resolution (understood as density) makes sense when talking about the reproduction of an object such as, for example, the image of a photograph or the scanning of a written or drawn page, but loses meaning when referred to a file totally generated by a vector graphics program.

The level of detail of a digital image depends on the resolution, that is, on the amount of information it contains. But to see the maximum resolution of the image, it is necessary that the reproduction system is also able to show it (Screen, printer, etc.) and that it has an adequate real size to show it to our eyes. To describe the various resolution densities, it is necessary to make some differentiations. Although similar, the resolution measurement units "DPI", "PPI" refer to separate measurement methods, although in practice they tend to be used even interchanging them.

DPI (Dots Per Inch): translation of "dots per inch". This unit of measure indicates the number of dots found in one inch. The term DPI is mainly used in the printing world, therefore refers to output devices such as inkjet or laser printers. It indicates the print definition and establishes the number of ink dots printed for each inch; consequently, the higher the DPI, the more detailed the image printed in that portion of the sheet will be.

PPI (Pixel Per Inch): translation of "pixels per inch". This unit of measure refers to input devices such as a camera or scanner, and represents the amount of pixels within an inch that a photographic sensor is able to scan and sample.

Screen resolution

The number of pixels on the monitor is defined by its resolution, it is expressed in the horizontal dimension and in the vertical one. Therefore, a monitor with a resolution of 1920 x 1080, has 1920 pixels in the horizontal dimension and 1080 in the vertical one. Basically, the resolution determines the image quality, the more pixels there will be, the more defined the image will be.

Main resolutions used in the computer and television fields:

Video Standard Resolution Aspect Ratio Color depth VGA 640x480 4:3 4 bpp SVGA 800x600 4:3 4 bpp XGA 1024x768 4:3 8 bpp HD 1280x720 16:9 24 bpp Full HD 1920x1080 16:9 24 bpp 4K 3840x2160 16:9 24 bpp 8K 7680x4320 16:9 30 bpp 16K 15360x8640 16:9 30 bpp

---

This is just an example of the various possible resolutions, they do not need to be learned by heart.

Some computers adopt completely different resolutions, for example on the macbookpro 2015, 15-inch, with combined Intel + Nvidia card, the screen resolution is:

2880x1800

On the same model, but with a 13-inch screen, the resolution is:

2560x1600

As you can see, neither of these two devices adopts a standard resolution, which is perfectly legitimate.

Color depth

Color depth indicates the number of bits used to indicate the color of a single pixel.

When referring to a pixel, the concept can be defined as the number of bits per pixel (bpp), which specifies the number of bits used.

Until the end of the 90s, with the lowest color depths, the value stored for each bit was generally an index in a color map or palette. The colors available in the palette itself can be determined by the hardware or modifiable.

It was customary to indicate the different types of depth with the name of the video standard with which they were introduced to the personal computer market.

- 1 bpp (2^1^ = 2 colors): monochromatic graphics, (in black and white)
- 2 bpp (2^2^ = 4 colors): CGA graphics
- 4 bpp (2^4^ = 16 colors): EGA or standard VGA graphics at low resolution
- 8 bpp (2^8^ = 256 colors): VGA graphics at high resolution, Super VGA

With the increase in the number of bits per pixel, the amount of possible colors also increases, making the use of palettes increasingly inconvenient. For the highest depths, it is therefore preferred to encode the colors directly in the values corresponding to the relative brightness of the red, green and blue channels according to the RGB model.

- 16 bpp (2^16^ = 65. 536 colors): High color
- 24 bpp (2^24^ = 16,777,216 colors): Truecolor

The truecolor depth model uses 24 bits and allows to reproduce images in a way very faithful to reality, coming to represent 16.8 million distinct colors. With this depth, 8 bits are used to represent red, 8 bits to represent blue and 8 bits to represent green. The 28 = 256 intensity levels for each channel combine to produce a total of 16,777,216 colors (256 × 256 × 256).

Below are 2 images: on the left we have an image with a color depth of 24 bpp, while on the right the same image with a color depth of 4 bpp.

Mostra immagine{width="8.391cm" height="4.48cm"} Mostra immagine{width="8.391cm" height="4.48cm"}Also for digital photographs, the same discussion made previously applies, in fact also photos, being digital images, are composed of a certain number of pixels.

In current cameras, the maximum resolution is often indicated in megapixels (MP). To make a concrete example, a camera that reaches 16 MP, with photos in 16:9 format will have a resolution of 5,312 pixels in width and 2,988 in height; multiplying these values, we get an area of 15,872,256 pixels, close to the nominal 16 MP.

Obviously, we indicate megapixels because we are talking about millions of pixels, if we hypothesize a resolution such as 50,000 x 50,000 we will have 2,500,000,000 pixels and therefore we could also indicate them as 2.5 gigapixels (GP).

Aspect Ratio

Another important characteristic is the aspect ratio, that is, the ratio between the width and height (always expressed in pixels) of the digital image.

This ratio also depends on the device used to represent the image: on some devices, the pixels have a rectangular shape, while if the aspect ratio of an image is modified, we will get a distortion, as we can see from the figure below:

Mostra immagine{width="8.961cm" height="5.039cm"} - Mostra immagine{width="7.16cm" height="5.39cm"}

For example, an image with a resolution of 1920 x 1080, therefore with an aspect ratio of 16:9 displayed on a monitor with an aspect ratio of 4:3 will appear distorted as well as horizontally compressed.

Mostra immagine{width="17cm" height="8.569cm"}

In the figure, on the left, five commonly used proportions (all represented with the same height). While on the right, adaptation of different image formats on TV/monitor with the same or different format.

Hexadecimal color code

This type of notation is called hexadecimal color code or **HTML code**.

Generally, in most graphic programs to unequivocally identify a very specific color, they are indicated with hexadecimal numbers. Hexadecimal numbers are numbers in base 16 and more precisely they are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, where zero is equal to 0 and F is equal to 15. The hexadecimal number with the highest value is FF which is equivalent to the decimal value 255.

Colors are indicated with a 6-character code preceded by the # symbol that indicates a color.

Color codes are hexadecimal triplets (and therefore a digit composed of 6 hexadecimal digits), which represent the colors red, green, and blue (the three primary colors).

The hexadecimal base is also used by various browsers for internet navigation, to unequivocally identify a very specific color.

Let's take, for example, the white color, in hexadecimal base it is indicated as `#FFFFFF` and is composed of:

- FF, i.e., Red = 255
- FF, i.e., Green = 255
- FF, i.e., Blue = 255

255 parts of red, 255 parts of green and 255 parts of blue

Below are some examples of colors with the corresponding hexadecimal code.

NAME CODE COLOR White `#FFFFFF`
Black `#000000`
Red `#FF0000`
Blue `#0000FF`
Green `#00FF00`
Yellow `#FFFF00`
Magenta `#FF00FF`
Cyan `#00FFFF`
Purple `#5C2E91`
Brown `#A56A1B`
Beige `#E3DAC9`
Azure `#0080FF`

Image compression

The following contribute to determining the size of the image:

- Resolution
- Color depth
- File format (Compressed or uncompressed, which in turn can be lossless or lossy)

The size in bytes of an image varies according to its resolution and according to the number of colors that a pixel can assume.

Assuming that in photo retouching and image processing techniques it makes little sense to decrease the color depth, to reduce the size of the image, it must be compressed, that is, with mathematical techniques, the number of bytes needed to contain the image must be reduced.

Example of file compression:

AAABBBBAAAAAAAAACCCC

(ASCII encoding weight 20 bytes)

the same sequence can be represented by

3A4B9A4C

(weight 8 bytes).

Graphics file format

Graphics includes a long list of different formats, because images can be encoded and/or represented in many different ways: moreover, many graphics programs that deal with images associate to the actual graphic data also a series of supplementary information, for example on their representation.

Graphic files have 2 main representations:

- raster: representation with a matrix of pixels, each with a color value
- vector: each element has a mathematically defined shape (straight line, circle, triangle, ellipse, parallelogram, etc.) in all its dimensions, and 2 coordinates that indicate its position.

The 2 formats are used for different purposes.

The vector format is typically used in the creation phase of drawings, from scratch, or when it is important to be able to enlarge or reduce (technical term: 'scale') the image without losing the quality of the same.

Moreover, with the vector format, the images have a very reduced size in terms of kB, or Mb.

The raster format is more used when working from pre-existing images, typically photographic, and you want to work on the color values of the image in general, or of an area of it, not going to manipulate the individual graphic elements. This type of format does not preserve the quality in scaling operations (enlargement and reduction).

With the raster format, the images have a very large size, in terms of kB or mB, compared to vector files, because each pixel of the image must be represented individually.

To remedy this drawback, raster images are almost always compressed.

Image formats with compression

There are many techniques for data compression and one must know how to distinguish between LOSSLESS and LOSSY techniques.

In the former, the space reduction does not involve loss of information and it is possible to return to the original format.

Some of these techniques are:

- **GIF**: (Graphic Interchange Format)
- **PNG**: (Portable Network Graphics)
- **TIFF**: (Tagged Information File Format)

LOSSY techniques, on the other hand, provide for a loss of the content of the image, but such as not to create loss of the meaning of the image itself.

The most widespread format is **JPEG**: (Joint Photographic Expert Group) widely used in the world of the internet where the complete image is not important but its meaning. However, it should be said that excessive compression causes an excessive loss of definition and color fidelity of the image.

Mostra immagine{width="17cm" height="10.343cm"}

Lossy compression is often to be preferred because it creates significantly more compressed images, while maintaining good quality.

However, there are situations in which using an approximated image is not acceptable:

- medical images
- historical documents
- images with legal value

Uncompressed image format

There are also formats for raster images that are not compressed, these file formats have minimal processing requirements, as compression algorithms are not necessary (in the writing phase) and decompression (in the reading phase), however, lacking compression, they are particularly voluminous, in terms of space occupied on disk (or other storage device), compared to other formats:

Some of these formats are:

- **BMP** (windows bitmap)
- **Raw** (raw image, there are many Raw files according to proprietary formats)

BMP format

Windows bitmap is a file format for digital images of bitmap type (also called raster type) used in digital graphics.

The ***.bmp** extension defines image files of this type (although less frequently the *.dib format is used)

One of the essential characteristics of the bitmap format, which has made it successful for a long time, is the speed with which the images are read or written to disk, much greater when compared to that of other types of files, especially on slower machines. In bitmaps, being uncompressed, the representation of data in RAM memory is largely similar, often identical, to that of data on disk: the processor is not forced to perform laborious calculations during coding and decoding operations and the data access time is often limited only by the hardware of the drive.

+--------------------+---+ | **DEFINITION:** | Uncompressed raster (non-vector) graphic format| +--------------------+---+ | **COMPRESSION:** | The BMP format does not use any type of | | | compression. | +--------------------+---+ | **APPLICABILITY:** | Medical images, such as for example those | | | produced by CT or MRI (Magnetic Resonance) are | | | encoded in a bitmap-type format. | | | | | Images that have a legal value, such as | | | copyright claim or computer forensics. | +--------------------+---+

In image processing, bitmaps prove to be suitable especially for the temporary storage of images that are modified often. Many scanning software for Windows save digitized images by default as bitmap files.

Digital sounds

The sound waves produced by the different sounds around us cause oscillations of pressure in very short times. The following figure shows the sound wave emitted by the human voice, recorded thanks to an audio management software.

The graph shows on the X-axis the time in seconds and on the Y-axis the gain in dB. We can notice how the waves of variation oscillate in very low times.

Mostra immagine{width="17cm" height="7.583cm"}

From a physical point of view, the sounds we hear are pressure waves of the air. The atmospheric pressure at sea level is about 1013 mbar, sounds are nothing but perturbations of the pressure characterized by very rapid variations.

Transducers are devices that transform pressure waves into oscillations of another kind. A typical example of an audio transducer is the microphone that "translates" the oscillations of the pressure into oscillations of electrical signal. Another example of transduction is represented by the undulations present in the grooves of vinyl records, which vary in a manner directly proportional to the pressure waves. Both these systems (microphone, vinyl record) use analog signals. Analog systems, despite being very faithful in recording, introduce considerable amounts of noise.

Sound sampling

To convert an analog signal to digital, sampling and quantization are performed. The analog signal, continuous in time and amplitude, is sampled, that is, measured at constant time intervals, obtaining a discrete signal in time and continuous in amplitude; then it is quantized, obtaining as a result a digital signal discrete both in time and amplitude.

The sound card of computers performs from 8,000 to 48,000 samplings per second (sampling frequencies from 8 to 48 kHz), then quantizing these data on 8, 10 or 32 bits.

In the case of sounds, we talk about temporal samples, while for images of spatial samples.

To reconstruct all the waveforms to be sampled in the most correct way possible, the sampling frequency should be at least twice these, as specified by the **Shannon theorem**. In the figure, a waveform (green) is shown which is sampled (black dots) at constant time instants.

Mostra immagine{width="12cm" height="3.461cm"}

Audio management software such as Audacity (which we deal with later) show waveforms that are sampled. By reducing more and more the scale of the X, where the time is indicated, we get to see the points of the individual samplings. The distance between one point and the next is about 0.0000227 seconds. Calculating the sampling frequency per second (1/0.0000227) we get in this case 44100 hertz.

Mostra immagine{width="11.141cm" height="6.249cm"}

Sampling consists of a series of measurements, each called a sample, similar to the sampling pixels for images. Each sample is represented in the computer's memory by a binary numerical value. The choice of the number of bits needed to represent the sound signal is very important to determine its precision and defines the bit depth, which determines the dynamic range of the signal and represents the ratio between the maximum and minimum values recorded in the time interval.

**Example of sampling at 2 and 4 bits: **the following graph shows sampling performed at 4 bits in the first half (on the left), then at 2 bits in the second half (on the right). The admissible values for the second sampling are 4 (2^2^) and are approximated with a certain imprecision. In the first half, where the precision bits are 4, with 16 admissible values (2^4^), the original wave is decidedly better approximated.

Mostra immagine{width="15.071cm" height="7.759cm"}

Guided exercise: estimating the speed of sound

In this example, we want to show how to calculate the speed of sound from an audio recording.

To perform the experiment, it is necessary to move the two earphones of the headphones as far apart as possible, pointing them both towards the same sound source. By producing a sound near one of the two earphones, the second earphone will record it with a certain delay, due to the distance between the two.

Mostra immagine{width="15cm" height="6.13cm"}

The speed of sound can be calculated since the two main data are known: the distance between the two earphones and the time taken by the sound to reach the second speaker. The estimate of the speed of sound will be better with the increase in the length of the cable that separates the two earphones of the headphones.

1.

Mostra immagine{width="5.62cm" height="10.389cm"}

1. In this case, having distanced the earphones about 1 meter from each other and having obtained in our experiment the result visible in the previous figure (0.0029 seconds) we can affirm that the speed of sound is:

${V = \frac{S}{t} = \frac{1m}{0.0029s} = 343}{m/s}$

1. "Algorithms"

Algorithms

An algorithm is a procedure that solves a given problem through a finite number of elementary steps.

In computer science, it can be represented graphically through flow diagrams, and formalized through a programming language to write a program.

The algorithm is a fundamental concept in computer science, because it is at the basis of the theoretical notion of computability, that is, a problem is said to be **computable** when it is solvable by means of an algorithm.

The algorithm is also a fundamental concept in the programming phase of software development: given a problem to automate, programming essentially constitutes the translation of an algorithm for such a problem into a program, written in a certain language, which can therefore be actually executed by a computer.

In order for an algorithm to be functional to the logic of programming, it must have the following characteristics:

- **Finite**: The algorithm must have a finite number of steps. It cannot go on forever.
- **Unambiguous**: The algorithm must be clear and unambiguous. Each instruction must have only one meaning and cannot be interpreted in different ways.
- **Deterministic**: The algorithm, given a certain input, must always produce the same output and always follow the same set of steps.
- **General**: The algorithm must be applicable to a set of similar problems, not just a single specific problem.
    - ***Example***: a specific problem might be "Determine the measurement of the hypotenuse of a right triangle whose catheti measure 5 cm and 7 cm", but developing an algorithm for this type of problem means finding a solution only for the right triangles that have the catheti of the dimensions of 5 and 7 cm. While instead a well-designed algorithm must be able to calculate the hypotenuse of a right triangle for all possible dimensions that the catheti can have, therefore the problem should be reformulated as "Determine the measurement of the hypotenuse of a right triangle, being known the measurements of the catheti."

Mostra immagine{width="17cm" height="10.991cm"}

To better understand, what is in practice an algorithm, here are some examples of algorithms in natural language:

Example of algorithms (outside of computer science):

Procedure that a professor uses to determine who is absent in a class.

1. Take the class register;

2. For each student in the list of the register, perform the following steps: a. Call the student; b. If the student responds "present", move on to the next student, otherwise write the name of the student on the register in the "absent" section;
3. Continue this process until each student in the list of the register has been called.

Procedure for cooking pasta

1. Fill a pot with water and put it on the fire;
2. Add a pinch of salt to the water;
3. When the water starts to boil, add the pasta;
4. Let the pasta cook for the time indicated on the package;
5. Turn off the stove and drain the pasta.

Mostra immagine{width="15.743cm" height="9.999cm"}

Procedure for shopping

1. Prepare a shopping list writing all the items you need;
2. Take a cart or a basket at the entrance of the supermarket and start walking along the corridors of the supermarket;
3. For each item on the list, look for the item in the supermarket;
4. When you find an item, put it in the cart or basket;
5. Repeat the 2 previous steps until you have found all the items on the list;
6. Once all the items have been found, go to the cash register to pay;
7. Put all the items in the shopping bags and take them home.

Examples of algorithms (computer science):

Swapping the value of two variables.

Let's suppose we have two variables, **a** and **b**, and we want to swap their values. Here's how the algorithm might appear:

1. Create a **temporary** variable and store the value of **a**.
2. Assign the value of **b** to **a**.
3. Assign the value of the **temporary** variable to **b**.

Finding the largest number in a list of numbers.

Let's suppose we have a list of numbers and we want to find the largest number. Here's how the algorithm might appear:

1. Set the first number in the list as the largest number so far.
2. For each number in the list, compare it with the largest number so far.

3. If the current number is larger than the largest number so far, then it becomes the new largest number.
4. Continue until the end of the list.

"Bubble sort" algorithm (bubble sort), sorting a list.

Let's suppose we have a list of numbers and we want to sort them in ascending order. Here's how the algorithm might appear:

1. Compare the first and second number in the list.
2. If the first number is greater than the second, swap them.
3. Move to the next pair of numbers (second and third) and repeat step 2.
4. Continue this process until the end of the list.
5. Repeat the entire process for the length of the list minus one.

Representation of an algorithm

In general, an algorithm can be represented in:

- Natural language
- Pseudocode
- Flow diagram
- Formal programming language

Natural language

The term "natural language" refers to the spoken or written language that people use daily to communicate with each other. Natural languages include languages such as Italian, English, French, German and all languages spoken by humans. These languages are rich in ambiguity, context and variety of meanings, which makes them understandable to humans, but often difficult to interpret for machines.

In the context of programming, the opposite of natural languages are "programming languages". Programming languages are formal languages created specifically to write computer code. These languages are designed to be precise, unambiguous and free of ambiguity, so that instructions can be executed by a computer without errors of interpretation.

Here are some key differences between a natural language and a programming language:

1. **Ambiguity**: Natural languages can be ambiguous and often depend on context to fully understand the meaning of words or phrases. Programming languages are unambiguous and require clarity and precision.
2. **Variation**: Natural languages vary widely between languages, dialects and registers. Programming languages are standardized and follow a specific syntax.
3. **Interpretation**: Natural languages are designed for communication between human beings. Programming languages are designed to be understood and executed by a computer.

4. **Purpose**: Natural languages are used to communicate ideas, thoughts and information between people. Programming languages are used to write precise instructions for a computer in order to perform specific operations.

For example, C++, Java, Python and other programming languages are examples of formal languages used to write computer code, while Italian, English and other human languages are examples of natural languages used for communication between people.

Take for example the algorithm for swapping values between two variables described above, it can be represented in natural language as follows:

1. Create a **temporary** variable and store the value of **a**.
2. Assign the value of **b** to **a**.
3. Assign the value of the **temporary** variable to **b**.

Flow diagram

A flow diagram (in English flow chart) is a graphical representation of a process or an algorithm in programming. It is a visualization technique used to represent in a clearly understandable way the flow of control within a program or a procedure. Flow diagrams are widely used in programming for design, documentation and communication purposes among the development team members.

A flow diagram helps programmers to visualize clearly the logic of an algorithm or a process, facilitating understanding, error correction and communication with other team members. It is a useful tool during the design phase and can be a valuable resource for code documentation.

These diagrams constitute the tool to present the reasoning that will lead to the expected result.

It allows to describe through a graphic modeling language:

- the operations to be performed, represented by conventional shapes (rectangles, diamonds, hexagons, parallelograms, rounded rectangles...), each with a precise logical meaning and inside which a textual indication describes the activity to be carried out;
- the sequence in which they must be performed, represented by connection arrows.

In algorithms created for the solution of mathematical problems, the instructions are developed through constants and variables linked by arithmetic operators (for example +, -, *, /, %), the result of which will be assigned to a variable.

Each diagram is characterized by a sequential reading:

- start from the initial block
- follow the outgoing arrow
- reach the next block and perform the operation described in the block
- perform all iterations until reaching the final block.

A flow diagram is composed of various symbols and arrows that represent different elements of an algorithm, including:

+----------+----------+ | ![| It is used to indicate the | |](Pictures/1000C77C00000AA600000 | starting point of the algorithm | | 8600BACD5B5.emf){width="2.619cm" | and inside it may appear the | | height="2.064cm"} | writings START or BEGIN. | | | | | | In a flow diagram only one is | | | present. | +----------+----------+ | ![| It is used for input operations | |](Pictures/100170FC00000FB700000 | or data writing and output or | | AA66CF76462.emf){width="3.889cm" | result reading. | | height="2.619cm"} | | | | It has only one incoming and | | | outgoing junction line. | +----------+----------+ | ![| It is used for calculation or | |](Pictures/10018F8C0000108A00000 | assignment operations and inside | | AF6B841A370.emf){width="4.075cm" | it is described in symbolic or | | height="2.699cm"} | literal form the operation to be | | | performed. | | | | | | It has only one incoming and | | | outgoing junction line. | +----------+----------+ | ![| It is used for comparison, | |](Pictures/1001C2AC0000145D00000 | control or choice operations | | A07BAE1709D.emf){width="5.027cm" | between alternatives to follow | | height="2.487cm"} | different sequences of | | | instructions depending on the | | | occurrence or not of events or | | | situations. | | | | | | Inside it is inserted the | | | expression to be considered, the | | | result of which is compared with | | | variables and constants through | | | the operators: | | | | | | - == (equal to) | | | | | | - > (greater) | | | | | | - >= (greater equal) | | | | | | - < (less) | | | | | | - <= (less equal) | | | | | | - <> (different) | | | | | | While the result is indicated on | | | the junction lines: | | | | | | - V (true) | | | | | | - F (false) | +----------+----------+ | ![| It is used to indicate the end | |](Pictures/1000D16C00000ADB00000 | of the algorithm and inside it | | 8AF8BEC7497.emf){width="2.672cm" | may appear the writings END or | | height="2.143cm"} | STOP. | | | | | | In a flow diagram only one is | | | present. | +----------+----------+

Moreover:

Mostra immagine{width="2.632cm" height="2cm"} Connect the various symbols to show the sequential or conditional path that the program follows.

Taking as an example the algorithm for swapping values between two variables described above, it can be represented with a flow diagram as follows:

Mostra immagine{width="6.278cm" height="15cm"}

Pseudocode

Pseudocode is a method of algorithmic representation that stands between natural language (such as Italian or English) and a formal programming language (such as Java, C++ or Python). It is a way of writing algorithms in a semi-structured form, using a combination of natural language and programming conventions, in order to describe a process or an algorithm in a clear and understandable way, without the need to adhere to a syntax of a specific programming language.

Pseudocode is particularly useful in the initial stages of designing an algorithm or a program, as it allows to express the basic idea without having to worry about the specific details of a programming language. This makes it a flexible tool for planning and communicating algorithmic ideas.

Obviously, there is no standard and conventionally used pseudo-language, in any case here are some common features that pseudocode should have:

1. Simplified language: Pseudocode uses a simplified language, similar to natural language, to describe steps and operations.

2. Informal syntax: It does not follow a rigorous syntax like a formal programming language. However, it often uses common conventions to represent concepts such as loops, decisions and assignments.
3. Focus on the idea: Pseudocode focuses on expressing the idea or algorithm without specific language details.
4. Ease of understanding: It is designed to be easily understood by programmers and non-programmers, facilitating the communication of programming logics.
5. Adaptability: It can be easily converted into real code in a specific programming language when moving from the design phase to the implementation phase.

Taking as an example the algorithm for swapping values between two variables described above, it can be represented in pseudocode as follows:

+----------+ | Begin | | | | Declare a, b, temp as integers | | | | a = 5 | | | | b = 10 | | | | Print "Before swap: a =", a, "b =", b | | | | temp = a | | | | a = b | | | | b = temp | | | | Print "After swap: a =", a, "b =", b | | | | End | +----------+

Programming languages

A programming language is a formal system designed to communicate instructions to a computer. They provide a set of rules and conventions that allow programmers to define algorithms and instructions in a format understandable to both humans and machines.

Programming languages play a fundamental role in software development and communication between programmers and computers.

The code written in a specific programming language is called **source code.**

A programming language has:

- **Key components:**
- **Syntax**: Rules that define how instructions must be written.
- **Semantics**: The meaning of instructions and how they are interpreted by the compiler or interpreter.
- **Typical features:**
- Variables and constants
- Arithmetic, relational and logical operations
- Control structures (conditionals, loops, etc.)
- Definition and calling of functions
- Input/output management

Classification of programming languages

A programming language can be classified:

1. Based on the level of abstraction:

- **Low level languages:** Closer to machine language, offer little or no level of abstraction. They have as characteristics direct control of hardware and architecture-specific code.
    - **Advantages**: Maximum efficiency and control.
    - **Disadvantages**: Difficult to write and maintain, not portable.
    - Examples:
        - **Assembly x86:** Used for low level programming on x86 processors.
        - **Assembly ARM:** Common in embedded and mobile device programming.
- **Medium level languages:** Provide some abstractions but maintain a certain control over hardware. Their characteristic is that they combine elements of high and low level.
    - **Advantages**: Good balance between control and abstraction.
    - **Disadvantages**: May require a deeper understanding of the underlying architecture.
    - **Examples:**
        - **C:** Widely used for the development of operating systems and system software.
- **High level languages:** Offer strong abstractions and are closer to human language. They have as a characteristic independence from hardware.
- **Advantages**: Easier to learn and use, greater productivity.
- **Disadvantages**: Less direct control over hardware, potentially less efficient.
- **Examples**:
    -
        - C++: Combines abstraction and control; offers object-oriented features while maintaining efficiency similar to C.
        - Java: Object-oriented; provides portability through the JVM and automatic memory management for greater productivity.
- **Very high level languages:** Offer a very high level of abstraction, approaching natural language or domain-specific notations. They have as characteristics a syntax very close to human language and are often oriented to solve problems in specific domains.
    - **Advantages: **Ease of learning and use, rapid prototyping and application development and reduction of programming errors
    - **Disadvantages: **May be less efficient in terms of performance, have limited control over low-level details and may not be suitable for all types of applications
    - **Examples**:
        - **Scratch**: Visual programming language for education. Uses graphic blocks to build programs, making programming accessible to beginners

        - **Python**: Known for its simplicity and readability, widely used in data science and web development.
        - **SQL (Structured Query Language):** Used for the management and interrogation of relational databases. Allows to express complex operations on data in a declarative way
        - **MATLAB**: Focused on numerical calculation and signal processing. Provides an integrated environment for mathematical calculations, visualization and programming

2. Based on the type of paradigm:

- **Imperative**: The imperative approach focuses on the sequential execution of instructions and the modification of the state of variables to obtain the desired result. Algorithms are expressed as a series of commands that modify variables and the flow of execution. (examples: C, Pascal)
- **Object-oriented:** It is a type of programming based on objects and classes, based on the concept of "objects", which are instances of classes. Objects contain data and methods that operate on such data. (examples: Java, C++, Python)
- **Declarative**: The declarative paradigm focuses on what should be done, without specifying how to do it. Algorithms are defined in a declarative way, providing the rules or conditions that guide the execution without implementation details. (examples: SQL)
- **Event-oriented: **It is a programming paradigm that is based on the management of events and responses to events. In this paradigm, the flow of the program is guided by the events that occur, such as mouse clicks, key presses, user input or system notifications. (examples: Visual Basic, JavaScript)

3. Based on the execution environment:

- **Compiled**: The source code is translated into machine language before execution. In particular, compiled languages have the following characteristics:
    - **Compilation phase**: In a compiled language, the source code is translated into machine language in a separate phase called "compilation". The result of this compilation is a platform-specific executable file.
    - **Execution:** After compilation, the executable program can be run directly without further translation steps. The operating system directly interprets the machine code contained in the executable.
    - **Performance**: Compiled programs tend to have higher performance than interpreted programs, as there is no need to translate the code during execution. The compiled executable can fully exploit the compiler's optimizations.
- **Interpreted**: The source code is executed line by line by an interpreter. In particular, interpreted languages have the following characteristics:
    - **Interpretation Phase:** In an interpreted language, the source code is not translated into machine language during the compilation phase. Instead, it is executed by an interpreter line by line or instruction by instruction during the program's execution.

- **Execution**: The interpreter reads and executes the source code directly. Each instruction is translated into machine code simultaneously with the running program.
- **Performance**: Interpreted programs tend to have lower performance than compiled programs because there is a greater consumption of resources associated with the interpretation instruction by instruction during execution.

4. Based on the approach to memory:

- **Manual memory management**: The programmer is responsible for memory management. (example: C or C++)
- **Automatic memory management**: The language automatically handles the allocation and deallocation of memory (examples: Java, Python).

Taking as an example the algorithm for swapping values between two variables, it can be implemented in various programming languages in the following ways:

+--+ | **Example in Java** | +--+ | public class Main { | | | | public static void main(String[] args) { | | | | int a = 5; | | | | int b = 10; | | | | System.out.println("Before swap: a = "+a+", b = "+b); | | | | int temp = a; | | | | a = b; | | | | b = temp; | | | | System.out.println("After swap: a = "+a+", b = "+b); | | | | } | | | | } | +--+ | | +--+ | **Example in C++** | +--+ | #include<iostream> | | | | using namespace std; | | | | int main() { | | | | int a = 5; | | | | int b = 10; | | | | cout << "Before swap: a = " << a << ", b = " << b << | | endl; | | | | int temp = a; | | | | a = b; | | | | b = temp; | | | | cout << "After swap: a = " << a << ", b = " << b << | | endl; | | | | return 0; | | | | } | +--+ | | +--+ | **Example in Python** | +--+ | a = 5 | | | | b = 10 | | | | print(f"Before swap: a = {a}, b = {b}") | | | | temp = a | | | | a = b | | | | b = temp | | | | print(f"After swap: a = {a}, b = {b}") | +--+

Program

The term "program" refers to a set of instructions written in a programming language that define a series of operations to be performed by a computer or a computer system. A program is fundamentally a detailed list of instructions that tell the computer what to do, step by step, in order to perform a specific activity or task.

A program can be very simple or extremely complex, depending on the needs of the application. For example, a program can be a simple script that calculates the sum of two numbers, or it can be a sophisticated operating system that manages all the functions of a computer.

With a computer we indicate both the tool for the representation and processing of information and an executor of algorithms.

Mostra immagine{width="17cm" height="7.137cm"}

Programs can interact with the external environment through input and output. Input can come from keyboards, mice, sensors or other devices, while output can be displayed on screens, printed on paper or sent through networks.

Difference between algorithm and program

The term program indicates the writing of an algorithm for a specific task, while an algorithm is defined as the description of the solution to a problem expressed through operations that the executor of the algorithm understands and can execute.

Programming depends on the task to be performed while an algorithm conveys the description of a solution to a given problem.

Programming

Programming means the activity with which the operations that serve to prepare the processor to perform a particular set of actions on particular data are defined, in order to solve a problem.

1. "Flow diagrams with Flowgorithm"

Introduction

Flowgorithm is a graphical programming tool that allows users to write and execute programs using flow diagrams. The flow diagram can be converted into different programming languages.

Creating a variable

A variable is an entity in which a value is stored. Variables are used to store data that must be used by the program.

A variable consists of three parts:

- **Name**: identifies the variable. It must be a valid identifier for the programming language in use.
- **Type**: indicates the type of data that can be stored in the variable. Data types can be numeric, strings or boolean.
- **Value**: the value stored in the variable. The value of a variable can be modified during the execution of the program

The creation of a variable occurs through its **declaration**, and is represented by the following block:

Mostra immagine{width="3.565cm" height="1.341cm"}

The **"Declaration"** block in its compilation is presented in this way:

Mostra immagine{width="11.432cm" height="8.999cm"}

Variable names (called identifiers) must follow the following rules:

- They must start with a letter.
- After the first letter, the identifier can contain letters or numbers.
- Spaces are not allowed.

A variable must have a name, and can contain values of four types:

- String (text);
- Integer (integer numbers);
- Real (numbers with a decimal point);
- Boolean (true or false, or more precisely **true** and **false**).

The declaration block also serves to create vectors (which will be illustrated later).

It is possible to declare multiple variables of the same type in the same block, entering the names separated by a comma.

Mostra immagine{width="13.653cm" height="12.899cm"}

If you need to create more variables of different types, you need to create a block for each type of variable, as in the example:

Mostra immagine{width="3.678cm" height="4.5cm"}

Assigning a value to a variable

To assign a value to a variable, you can use two blocks, that of **reading** or of **assignment**:

Mostra immagine{width="7.692cm" height="1.235cm"}

In the **"Reading"** block (or keyboard input), the value to the variable will be assigned during the execution of the program, and therefore in a certain sense its value will be unknown during the construction of the flow diagram.

In the **"Assignment"** block, the value to the variable will be defined during the creation of the flow diagram and can contain **values** or **mathematical expressions**.

The **"Assignment"** block in its compilation is presented in this way:

Mostra immagine{width="12cm" height="7.525cm"}

Arithmetic operators

The arithmetic operators used to define expressions or perform calculations are the following:

+ addition **-** subtraction ***** multiplication **/** division **%** modulo (returns the remainder of a division)

Printing a text and/or the value of a variable on screen

To print a text or a variable on screen, it is necessary to use the **"Writing"** block:

Mostra immagine{width="3.493cm" height="1.235cm"}

Inside the writing block, all the **text** that you want to print on screen must be put **"in quotation marks"** (called double quotes) so that it is recognized as a string (that is, as a text), while if you want to print the value of a **variable** on screen, you must enter the name of the variable **without the quotation marks**.

If, on the other hand, inside the "Writing" block you want to insert both a text and the value of a variable, you can use the concatenation operator **&**.

Mostra immagine{width="12cm" height="16.198cm"}

Joining multiple variables

The union between variables and compatible formats is called "concatenation", and we must make the distinction between numeric variables (such as integers and/or reals that are compatible with each other) and variables of string type (where even numbers are treated as characters).

Union between numeric type variables

If you want to make a sum between numeric variables (integer and/or real) you must use the "+" operator (addition).

So in the "assignment" block you can find these 2 cases:

- variable = variable+NUMBER
- variable = variable1+variable2

Mostra immagine{width="15.051cm" height="11cm"}

Obviously, between numeric type variables it is possible to perform all arithmetic operations (addition, subtraction, multiplication, division and remainder) using the arithmetic operators seen previously (+ - * / %).

Union between string type variables

If you want to make a union between string type variables you must use the "&" operator (concatenates).

- variable = variable&"TEXT"
- variable = variable1&variable2

Mostra immagine{width="15.051cm" height="11cm"}

Condition

The "Condition" block controls a boolean expression and passes to the True or False branch depending on whether the condition is verified or not.

Mostra immagine{width="3.986cm" height="2.295cm"}

In the **"Condition"** block a comparison between two variables (or between a variable and a value, or even between the results of expressions) must be inserted, if this comparison is satisfied then the flow will follow the **"True"** branch, otherwise it will follow the **"False"** branch.

Mostra immagine{width="12cm" height="7.772cm"}

Relational operators

To make a comparison, relational operators are used, which are the following:

> greater than	**>=** greater or equal to	**<** less than	**<=** less than or equal to	**==** equal to	**!=** not equal to		

For example, if you want to verify that the value of a variable is positive or not, it can be schematized as follows:

Note: It is not said that the "False" side branch must necessarily follow the execution of some block:

Mostra immagine{width="7.908cm" height="10.201cm"}

In case you want to verify more than one condition to reach the resolution of an algorithm, you can follow two paths:

- branch the conditions
- insert, if possible, all the conditions inside a single block, through the logical operators.

Branched conditions

For example, suppose you want to verify that an integer X in input is simultaneously both positive and even:

Mostra immagine{width="16.725cm" height="14cm"}

Another example, verify that an integer X is simultaneously divisible by 2, by 3 and by 5:

Mostra immagine{width="13.474cm" height="14cm"}

Logical operators

It is possible to avoid the branching of conditional constructs through logical operators.

- **&&, **(which means AND), that is, both conditions must be satisfied;
- **|| **(which means OR), that is, only 1 condition must be satisfied.

In practice, in a condition block, it is possible to insert as a conditional expression (condition1)&&(condition2)&&(condition3), which means that all three conditions must be verified in order to pass to the "True" branch of the condition.

In (condition1), (condition2) and (condition3) comparison expressions must be inserted such as x>0 or x%2==0.

Example, taking into consideration the previous example, that is, suppose you want to verify that an integer X in input is simultaneously both positive and even, this becomes:

Mostra immagine{width="9.43cm" height="9.999cm"}

Instead, taking into consideration the example of verifying that an integer X is simultaneously divisible by 2, by 3 and by 5, it becomes:

Mostra immagine{width="9.195cm" height="9.999cm"}

To understand instead how to use the logical operator "OR" (which is indicated with the double vertical bar "||" which can be written with the combination of SHIFT + "\" keys, next to the "1" key) one can help with an example.

Given an integer X in input, verify if it is equal to 3 or if it is equal to 5 writing in output "OK", while if it is different print in output "Not verified".

You can use a nesting of conditions as in the figure:

Mostra immagine{width="9.978cm" height="9.5cm"}

Or simply:

Mostra immagine{width="8.915cm" height="9.5cm"}

1. "Fundamentals of programming in Python"

Introduction to the Python language

Python is a high-level, interpreted, and object-oriented programming language, originally developed by Guido van Rossum in 1991. It is one of the most popular programming languages in the world, used to create a wide range of applications.

Python is known for its clear and readable syntax, which emphasizes code readability and reduces the cost of software maintenance. It is a multi-paradigm language that supports object-oriented, imperative, and functional programming.

This language is widely used in many fields, including web development, data analysis, artificial intelligence, machine learning, system script development, and many other areas where ease of use and productivity are crucial.

Features of Python

Python has a series of features that make it a versatile and powerful programming language:

- **Simplicity and Readability:** Python's syntax is designed to be clear and intuitive, making the code easy to read and write.
- **Object-Oriented Programming (OOP):** Python fully supports OOP concepts, including encapsulation, inheritance, and polymorphism, allowing developers to create modular and reusable programs.
- **Automatic Memory Management:** Python uses a *garbage collector*, which automatically manages memory, freeing developers from this task and reducing the risks of memory management errors.
- **Dynamic Typing:** Python is a dynamically typed language, which means that it is not necessary to declare the type of a variable before using it.
- **Vast Standard Library:** Python comes with a rich standard library that offers tools for a wide range of tasks, from string processing to network programming.
- **Extensibility:** Python can be extended with modules written in other languages such as C or C++, allowing to optimize the critical parts for performance.

Conventions

Before starting to write code in Python, it is good to learn some conventions that form the basis of clean and easy-to-understand code writing:

- **Meaningful names for variables and functions:** Using names that reflect the purpose of the variable or function can make the code more readable and maintainable.
- **PEP 8:** Python has an official style guide called PEP 8, which provides guidelines for code formatting. Following these guidelines helps maintain consistency across Python projects.

- **Indentation:** In Python, indentation is semantically significant and determines the structure of the code. In other words, Python uses indentation to define code blocks. This is fundamental when writing functions, loops, or conditional structures. It is ***fundamental*** to maintain consistent indentation.
- **Comments and documentation:** Commenting the code and maintaining updated documentation are essential practices. Python supports docstrings, which are documentation strings embedded in the code.
- **Encapsulation Principle:** Although Python does not have private access modifiers like other languages, the convention of prefixing with an underscore the names of attributes and methods that should be considered private is used.
- **Avoiding excessive use of globals:** Excessive use of global variables can make the code difficult to debug and maintain. It is preferable to limit their use.
- **Structure of constant names:** In Python, constants are typically written in capital letters, with words separated by underscore, such as: PI_GREEK.

By following these conventions and leveraging the powerful features of Python, developers can create efficient, readable, and maintainable code.

Guide to Python development environments

An IDE (Integrated Development Environment) is a development environment that facilitates programming. Below are presented the main development environments available for Python, with their distinctive features.

Main IDEs for Python

Below are listed some IDEs for programming in Python

PyCharm Community Edition

PyCharm Community Edition is a professional development environment that offers:

- Intelligent automatic code completion
- Syntax highlighting
- Integrated debugger
- User-friendly interface
- Free version with full features

Visual Studio Code

Visual Studio Code with the Python extension is characterized by:

- Lightness and execution speed
- High customizability
- Optimized support for Python
- Vast user community

- Free and open source nature

Thonny

Thonny is an environment particularly suitable for beginners, which presents:

- Simplified and intuitive interface
- Installation including Python
- Visual debugger with step-by-step execution
- Optimized configuration for learning

Standard IDE

Called 'IDLE' (Integrated Development **and Learning** Environment) included with Python offers:

- Automatic installation with Python
- Basic but complete features
- Adequacy for simple projects
- Absence of need for additional configurations

Guide to Installing Thonny

To start the installation of Thonny it is necessary to:

- Access the official site: https://thonny.org
- Select the appropriate download for the operating system in use

Installation on Windows

The installation procedure on Windows includes:

- Execution of the downloaded .exe file
- Follow the guided installation procedure
- Confirm the steps until completion
- Wait for the automatic start of the application

Installation on MacOS

For MacOS, the process requires:

- Opening the downloaded .pkg file
- Follow the installer's instructions
- Locate the application in the Applications folder
- Configure any security permissions at the first start

Work Environment

The Thonny interface consists of:

- Text editor in the upper section for code writing
- Python shell in the lower section for viewing results
- Toolbar with the main commands

Mostra immagine{width="17cm" height="9.246cm"}

Syntax in Python

The syntax in Python is a set of rules that define how to correctly write code so that it can be understood and executed by the Python interpreter. Here is an overview of the fundamental syntax rules in Python:

Basic Structure of a Program

A typical Python program does not require a specific structure as in C++ or Java. However, here is an example of a common structure:

+---+ | # Import of libraries | | | | import math | | | | # Definition of functions | | | | def greet(name): | | | | print(f"Hello, {name}!") | | | | # Main body of the program | | | | if __name__ == "__main__": | | | | greet("World") | | | | print("The value of pi is about", math.pi) | +---+

- **import math** - This instruction imports the math module, which provides mathematical functions.
- **def greet(name): **Defines a function called greet.
- **if __name__ == "__main__":** This block is executed only if the script is run directly (not imported as a module).

Moreover, it is fundamental to remember that unlike C, C++ and Java languages:

- Strings in Python can be enclosed in single or double quotes (example: 'Hello' or "Hello").
- Python uses indentation to define code blocks, not curly braces.
- Instructions in Python do not require a semicolon at the end.

Editor, interpretation and execution of a program

You can write Python code with any simple text editor. However, for a better programming experience, it is recommended to use a specific IDE for Python, such as PyCharm, Visual Studio Code with Python extensions, IDLE (which comes with Python), Spyder, or Jupyter Notebook, which offer syntax highlighting, code suggestions and other useful features.

File Extension: Python files are saved with the *.py extension.

Execution: Python is an interpreted language, so there is no separate compilation phase. To run a Python program, you need to open the terminal or command prompt and go to the directory where the file was saved (for example test.py). Then, run the command:

---

python test.py

This command tells the Python interpreter to execute the test.py script.

In case a specific IDE is installed, the program after the initial configuration, will execute the Python script through specific functions without going through the command prompt.

Python also offers an interactive mode, accessible by simply typing python in the terminal, which allows to execute Python instructions one at a time, useful for quick tests and learning.

Example program in Python

+--+ | # This is a single line comment | | | """ | | | This is a comment | | | on multiple lines | | | """ | | | print("Hello everyone!!!") # Prints "Hello everyone!!!" on the | | screen | +--+

Explanation of the Program

- **Comments**: In Python, single line comments start with #. For comments on multiple lines, you can use three quotes """ at the beginning and end of the comment block.
- **print("Hello everyone!!!")** - This built-in function prints the message "Hello everyone!!!" on the screen. Python provides basic functions like print() without the need to import additional libraries as happens in other programming languages.
- **Program structure:** Python executes the instructions in the order in which they appear, from top to bottom. An explicit main function for the execution of the program is not necessary as happens in other programming languages.
- **Syntax of instructions:** In Python, instructions do not require a special termination character. Each new line is interpreted as a new instruction.

Variables

Variables in Python are dynamic references to objects in memory. The object is the actual area of memory that contains the data.

In Python, variable names must follow these rules:

- Variable names in Python must be a continuous sequence of characters, without spaces. (so for example "count even" must become "count_even" or "countEven");
- They must start with a letter (a-z, A-Z);
- Start with underscore for private variables: _private_variable;
- The rest of the name can contain letters, numbers and underscores.
- Names are case-sensitive (variablePersonal and variablepersonal are different variables).
- They cannot be Python keywords (such as if, for, while, etc.).

Examples of valid variables in Python:

+----------------------+ | my_variable = 5 | | | | count = 0 | | | | _private = "Hello" | | | | camelCase = True | +----------------------+

Declaration and initialization of variables

Python is a dynamically typed language, which means that it is not necessary to explicitly declare the type of a variable. The type is automatically inferred when a value is assigned. Here are some examples:

+---+ | x = 5 # x is automatically an integer | | | | y = "Hello" # y is automatically a string | | | | z = 3.14 # z is automatically a float | +---+

Scope of variables

- Global variables: declared outside of any function
- Local variables: declared inside a function
- Use of the global keyword to modify global variables inside functions

Examples:

+---+ | x = 10 # Global variable | | | | def function(): | | | | global x | | | | x = 20 # Modifies the global variable | | | | y = 5 # Local variable | +---+

Deletion of variables

To delete a variable, use the del command, such as:

---

del name # Removes the variable 'name'

Types of Variables

Python is a dynamically typed language, but strongly typed.

Integer numbers (int)

Arbitrary precision (there is no limit to the size):

+-------------------------------+ | a = 5 | | | | b = 12345678901234567890123 45 | +-------------------------------+

Decimal numbers (float)

They follow the IEEE 754 double precision standard:

+-------------------------------------+ | x = 3.16 | | | | y = -0.5 | | | | z = 2.5e-4 # Scientific notation | +-------------------------------------+

Complex numbers

A complex number is a number of the form ***z = a + bi***, where ***a*** and ***b*** are real numbers and ***i*** *is the imaginary unit. such that* ***i^2^ = **-1***, where ***a*** is the real part, ***b*** the imaginary part and ***i*** is the imaginary unit.

---

c = 3 + 4j # j represents the imaginary part

Strings (str)

They are immutable and support indexing and slicing

+----------+ | text = "Computer Science" | | | | print(text[0]) # Output: 'C' | | | | print(text[1:4]) # Output: 'omp' | +----------+

On a string it is possible to use some methods (or functions) to simplify manipulation:

+----------+ | s = "hello, world" | | | | print(s.capitalize()) # Output: 'Hello, world' | | | | print(s.upper()) # Output: 'HELLO, WORLD' | | | | print(s.find('o')) # Output: 4 | | | | print(s.split(',')) # Output: ['hello', ' world'] | +----------+

Boolean Variables (bool)

They are variables that can only assume 2 values: True and False (note the capital letter).

+----------+ | x = True | | | | y = False | +----------+

Simple operators

Naturally, to perform operations between variables, "special" characters must be used, which are called operators.

Operators can be divided into various categories. Below are listed the main operators and those most commonly used.

In Python, operators play a crucial role in allowing operations to be performed on variables and values. They are divided into different categories, each with its own purpose. Here is a summary of the simple operators in the specified categories: relational, arithmetic, logical, and assignment.

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations such as additions, subtractions, multiplications, divisions, and obtaining the remainder of a division (modulo).

+ addition

- subtraction

* multiplication

/ division (always float result)

// integer division (rounds down)

% modulo (remainder of the division)

** Exponent (power)

Relational Operators

Relational operators compare two values and return a boolean value (True or False).

> greater than

>= greater than or equal to

< less than

<= less than or equal to

== equal to

!= not equal to (different from)

Logical Operators

Logical operators are used to combine two or more conditions.

and logical and

or logical or

not logical not

Assignment Operators

Assignment operators are used to assign a value to a variable.

The assignment operation normally allows to assign different values to the same variable during the execution of the program.

The basic assignment is executed through the "=" operator which, in a very simple way means "take the value on the right side and copy it to the left side".

The basic operator is =, but there are also compound assignment operators that combine an arithmetic operation with the assignment.

+-------+--+ | = | **Assignment, assigns a certain value to a variable.** | | | | | Example x = 2 or x = y | +-------
+--+ | += | **Assignment with addition**