PARTE 1

Concetti informatici fondamentali

Nicolò Bartolini

Argomenti

Cos’è un computer (tecnicamente)

Digitalizzazione delle informazioni

Linguaggi di programmazione

Conclusioni

→ **Cos’è un computer (tecnicamente)**

Digitalizzazione delle informazioni

Linguaggi di programmazione

Conclusioni

Cos'è un computer (tecnicamente)

Un **computer** è un dispositivo elettronico in grado di svolgere **operazioni matematiche** e logiche e di memorizzare informazioni.

Ciò che distingue un computer dal cervello umano è la **velocità** con cui i computer effettuano le operazioni matematiche. In altre parole, i computer odierni riescono ad eseguire **miliardi di istruzioni al secondo**.

Queste istruzioni consistono in semplici calcoli matematici (addizioni, sottrazioni, ecc.). Nonostante la loro semplicità, la velocità con cui vengono eseguite permette a tali operazioni di far funzionare **tutta l'informatica che utilizziamo quotidianamente**.

Ciò che distingue, invece, un computer da una semplice calcolatrice (almeno quelle più semplici) è la capacità dei computer di essere **programmabili**.

Quindi anche gli smartphone, i tablet, i dispositivi smart home, alcuni elettrodomestici e altri oggetti sono considerati computer. Tuttavia, per semplicità, ci concentreremo solo su quelli "tradizionali".

I processori dei computer sono caratterizzati da una velocità espressa in GHz (Giga Hertz), che indica esattamente il numero di operazioni che riescono ad eseguire in un secondo.

Architettura dei computer

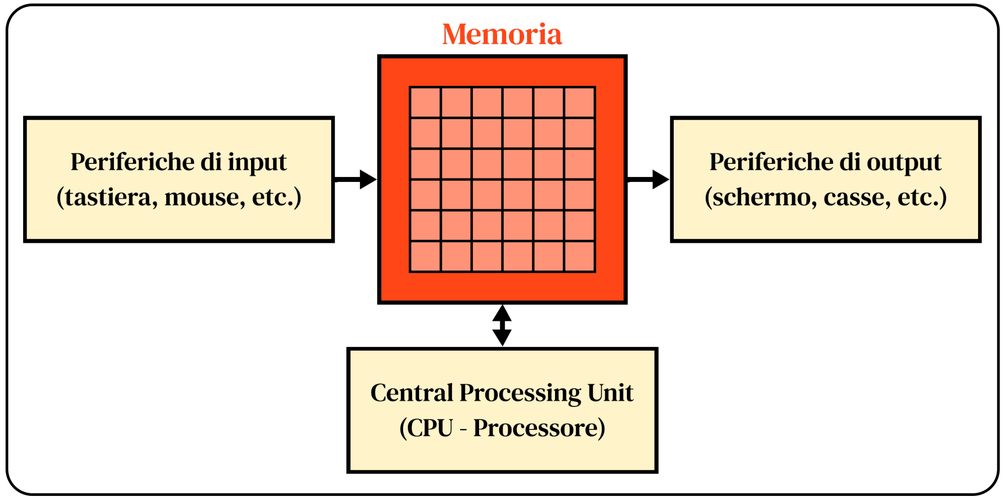
Tutti i computer sono caratterizzati da una struttura standard, chiamata **architettura di von Neumann**, ideata dal matematico e fisico John von Neumann negli anni ‘40.

il bro

Secondo l’architettura di von Neumann, come si vede dalla figura, i computer sono costituiti da degli elementi fondamentali: degli strumenti per fornire **informazioni**, un elemento per memorizzare le **informazioni**, un componente per elaborare le **informazioni** e degli strumenti per visualizzare le **informazioni**.

Tutte queste informazioni, però, per poter essere elaborate dal computer, devono, in qualche modo, essere convertite in un **formato** che il computer possa **comprendere**.

In altre parole, le informazioni devono essere **digitalizzate**.



Cos’è un computer (tecnicamente)

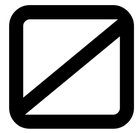
→ **Digitalizzazione delle informazioni**

Linguaggi di programmazione

Conclusioni

Il codice binario

Per far sì che il computer possa lavorare con le informazioni, dunque, è necessario **digitalizzarle**. La digitalizzazione delle informazioni si basa sui famosi:

 e 1

Il **codice binario** è semplicemente un modo per rappresentare i numeri. Noi utilizziamo il codice decimale, ovvero la **base 10**, il computer usa il codice binario, ovvero la **base 2**.

Il funzionamento del codice binario è più semplice di quanto sembri: così come **noi utilizziamo 10 simboli** per rappresentare i numeri (**0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**), **il codice binario utilizza 2 simboli** (**0** e **1**).
Noi contiamo **facendo seguire i 10 simboli che abbiamo a disposizione** e, dopo averli finiti ne fissiamo uno a sinistra e li facciamo seguire a destra. Il codice binario funziona nello stesso modo, ma con solo 2 simboli.

1 -> 2 -> 3 -> ... -> 9 -> **1**0 -> **1**1 -> **1**2 -> ... -> **9**9 -> **10**0 -> **10**1 -> ...

0 -> 1 -> **1**0 -> **1**1 -> **10**0 -> **10**1 -> **11**0 -> **11**1 -> **100**0 -> **100**1 -> **101**0 -> **101**1 -> **110**0 -> **110**1 -> **111**1 -> ...

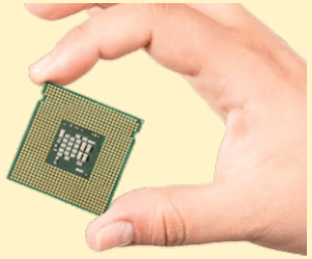
https://www.youtube.com/watch?v=zELAfmp3fXY

Perché il codice binario?

La ragione principale per cui i computer utilizzano il codice binario risiede nelle **comodità tecnologiche**.

Il **processore**, il cervello di un computer, è un **circuito elettrico** costruito con numerosi **transistor**. I transistor sono **minuscoli componenti elettronici** che, grazie alle proprietà chimiche del **Silicio**, possono essere attivati o disattivati rapidamente. Quando un transistor è **attivo**, ha un valore specifico di **tensione elettrica**; quando è **spento**, la tensione è **nulla**. Quindi, **un transistor attivo rappresenta il numero 1, mentre uno spento rappresenta il numero 0**.

Un processore moderno delle dimensioni di qualche centimetro è capace di contenere decine di miliardi di transistor.



L'utilizzo del codice binario è vantaggioso perché **è più semplice per il computer** capire che **il valore è 1 quando c'è elettricità e 0 quando non c'è**, piuttosto che associare 10 diversi valori di tensione elettrica a valori numerici diversi.

Il secondo motivo, derivante dal precedente, è che con soli due simboli è **meno probabile commettere errori**. Ogni cifra di un numero composto da molte cifre sarà 0 o 1. Se usassimo il codice decimale, ogni cifra potrebbe avere 10 valori diversi, aumentando la probabilità di errore.

https://www.instagram.com/p/C_LFZyxBI9z/

Numeri ricorrenti nell'ambito informatico

Il codice binario è caratterizzato da due simboli, e in informatica ci sono numeri che ricorrono spesso, come le **potenze del numero 2**.

2^0 = 1; 2^1 = 2; 2^2 = 4; 2^3 = 8; 2^4 = 16; 2^5 = 32; 2^6 = 64; 2^7 = 128; ...; 2^10 = 1024; 2^11 = 2048; ...

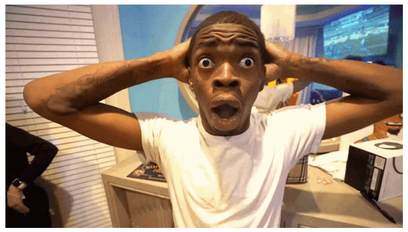
Una potenza del 2 significativa è **2^8 = 256**. In questo caso, la **base "2"** rappresenta il numero di cifre utilizzate per digitalizzare l'informazione (0 e 1), l'**esponente "8"** indica il numero di **slot** per posizionare le cifre, e il **risultato "256"** corrisponde alle possibili combinazioni ottenute posizionando le cifre negli otto slot.

Ogni slot è chiamato **bit**. Gli 8 slot menzionati (quindi **8 bit**) equivalgono a **1 Byte**.

8 bit = 1 Byte [B]

Le altre potenze del due rilevanti entrano in gioco qui. Ad esempio, **2^10 = 1024 Byte** corrispondono a **1 KiB**, ossia un **KibiByte**...

...esatto, non kiloByte, ma KibiByte ⟶



⟶

Il prefisso "kilo" è utilizzato in ambito scientifico e rappresenta esattamente 1000 dell'unità di misura associata. Quindi, 1 KB equivale a 1000 Byte, mentre 1 KiB equivale a 1024 Byte, così come 1 MiB (MibiByte) equivale a 1024 KiB, e così via...

Digitalizzazione di testi, immagini e video

Abbiamo affrontato la digitalizzazione dei numeri senza entrare troppo nei dettagli tecnici. Ora, con lo stesso approccio, esaminiamo brevemente il processo di digitalizzazione di testi, immagini e video.

Spoiler: ogni informazione è rappresentata come una sequenza di numeri binari.

Digitalizzazione dei testi

I **testi** sono essenzialmente una **serie di caratteri**. Ad esempio, la parola “ciao” è composta dai caratteri “c”, “i”, “a” e “o”. Pertanto, digitalizzare un testo significa **assegnare un numero a ciascun carattere**.

Per fare ciò, esistono diversi **standard**. Il primo ad essere stato sviluppato è lo standard **ASCII** (1986). Come tutti gli standard, mirava a **ottimizzare** l'uso della memoria, cercando di utilizzare il **minor numero di bit possibile** per rappresentare ogni carattere. ASCII utilizza **7 bit**, permettendo così di rappresentare 2^7=**128 caratteri diversi**. Anche se 128 potrebbero sembrare molti, in realtà sono **pochi** considerando che devono includere lettere maiuscole, minuscole, simboli e vari caratteri di controllo. Quindi, sebbene ASCII sia efficace per rappresentare i caratteri di base, non è sufficiente per caratteri speciali come lettere accentate o emoji.

Per affrontare questa limitazione, è stato introdotto lo standard **UNICODE**, un sistema a **32 bit** che consente di rappresentare **oltre 2 miliardi di caratteri** (2^32). Spesso quei bit vengono inutilizzati, quindi per ridurre lo spazio occupato si utilizza la **transcodifica UTF-8**.

Digitalizzazione delle immagini

Come tutti sappiamo, le immagini sono costituite dai pixel. Ma cosa sono i pixel?
Un pixel è un quadratino colorato che rappresenta un mattoncino di un'immagine. In altre parole, le immagini digitali non sono altro che mosaici di pixel.
Da ciò è immediato comprendere che la digitalizzazione delle immagini avviene associando ad ogni pixel uno o più numeri che rappresentano il colore di quest'ultimo e indicando in che modo disporre questi pixel.
Più tecnicamente, le immagini a colori vengono rappresentate attraverso l'assegnazione di 3 Byte per ogni pixel. Ognuno di questi Byte (8 bit) è un numero binario compreso tra 0 e 255 (11111111) che indica rispettivamente l'intensità del rosso, del verde e del blu. Da qui deriva la nomenclatura RGB (Red Green Blue).

Digitalizzazione dei video

Cosa sono i video se non una sequenza di immagini?

Da ciò è immediato comprendere che la digitalizzazione dei video avviene digitalizzando tutte le immagini che lo compongono.

Dal concetto di digitalizzazione dei video deriva il concetto di FPS (frames per second, fotogrammi per secondo), che chi fa del GAMING conosce bene.
Gli FPS non sono altro che il numero di immagini che si susseguono in un secondo di durata di un video. Quindi, in un video che "viaggia" a 60 fps ci sono 60 immagini in ogni secondo.
Fun fact: i film sono quasi sempre registrati a 24 fps.

Cos’è un computer (tecnicamente)

Digitalizzazione delle informazioni

→ **Linguaggi di programmazione**

Conclusioni

Cos'è un programma

Abbiamo esaminato rapidamente il processo di digitalizzazione delle informazioni nei computer. Riprendendo l'introduzione effettuata nella slide 4, i computer che usiamo quotidianamente si basano sull'esecuzione di **miliardi di istruzioni al secondo**. Ma da dove provengono queste istruzioni?

Un programma consiste in una sequenza di istruzioni che il computer deve eseguire.

Pertanto, quando un computer **esegue istruzioni**, segue il **flusso** impartito da un **programma** specifico.

Quando accendete il computer e si avvia il sistema operativo (i.e.: Windows), state, in realtà, avviando un programma (il sistema operativo), che a sua volta fa avviare tanti altri programmi, tra cui, ad esempio, l'esplora risorse, il software che gestisce gli aggiornamenti, e molto altro.

Di conseguenza, il termine "programmare" significa esattamente "fornire al computer una sequenza di istruzioni".

Sorge spontaneo chiedersi, dato che il computer ragiona in **binario**, come sia possibile impartire istruzioni. Si potrebbe pensare che sia necessario specificare le operazioni matematiche di base che il computer deve eseguire sui numeri binari. Ovviamente, ciò non è necessario (anche perché sarebbe terribile). Per risolvere questo problema, entrano in gioco i **linguaggi di programmazione**, che consentono di **comunicare** al computer le **istruzioni da eseguire**. Un altro programma apposito **tradurrà** poi le istruzioni specificate nel linguaggio di programmazione in operazioni matematiche di base binarie, comprensibili al computer.

Il linguaggio dei computer

I computer ragionano **diversamente** rispetto a come ragioniamo noi. I computer sono **rigorosi** e **formali** e, per questo motivo, è necessario utilizzare un **linguaggio appropriato** per impartire loro istruzioni, ovvero un **linguaggio di programmazione**.

Siamo abituati a comunicare sfruttando il **linguaggio naturale** (i.e.: italiano, inglese, ecc.) mentre per impartire istruzioni agli elaboratori dobbiamo usare **linguaggi formali**.

Tutti i linguaggi di programmazione sono linguaggi formali.

Linguaggio naturale

"La vecchia porta la sbarra"

"Hai visto l'ultima live di Dario Moccia? Mio padre totale. Ultimamente fa morire dal ridere."

Il **linguaggio naturale** è ricco di riferimenti esterni, modi di dire e metafore, ed è **soggetto a interpretazione**: una condizione **non ideale** per la comunicazione con una macchina.

Linguaggio formale

"x = 10"

"print("Hello, world!")"

Il **linguaggio formale**, invece, **non è soggetto a interpretazione**: ha regole molto rigide che definiscono il significato di ogni "parola" (**keyword**, **istruzione**). Le macchine possono leggerlo (**interpretarlo**, **compilarlo**) con precisione.

Immagina di inserire dentro un contenitore che chiamiamo x un valore intero positivo e più specificamente il numero 10 in decimale, salvane il valore in memoria per poterlo recuperare successivamente. grazie.

$x = 10$

Problemi, algoritmi e programmi

Abbiamo capito che programmare significa **impartire istruzioni alla macchina utilizzando un linguaggio formale**. Ma, per quale motivo vorremmo farlo? La risposta è per **risolvere problemi**.

Algoritmo

Un **algoritmo** è l'insieme delle **operazioni necessarie** per risolvere un problema (i.e.: le istruzioni per costruire un LEGO®).

Dato uno specifico problema, possono esistere più algoritmi che permettono di risolverlo.

Programma

Sappiamo che un programma è una **sequenza di istruzioni** che devono essere eseguite da un computer. Quindi, più tecnicamente, un **programma** è l'**implementazione** di un algoritmo in un qualsiasi **linguaggio formale**.

Un algoritmo possiede infinite implementazioni in diversi programmi (nello stesso e in altri linguaggi formali). Se un problema è risolvibile in un modo, allora è risolvibile in infiniti modi.

Esempi

① PROBLEMA: È il compleanno del mio amico Davide (ad agosto) e voglio mandargli una cartolina.

ALGORITMO:

1. Trovare un negozio che vende cartoline aperto ad agosto.
2. Andare nel negozio che vende cartoline.
3. Comprare la cartolina.
4. Compilare la cartolina con i dati di Davide e una frase di auguri.
5. Spedire la cartolina.

② PROBLEMA: Effettuare l'addizione di due numeri scelti dall'utente.

ALGORITMO:

1. Chiedere il primo addendo all'utente.
2. Memorizzarlo.
3. Chiedere il secondo addendo all'utente.
4. Memorizzarlo.
5. Effettuare l'addizione tra i due numeri memorizzati.
6. Mostrare il risultato all'utente.

Osserviamo che l'algoritmo non specifica minimamente dettagli tecnici o tecnologici. L'algoritmo, infatti, rappresenta esclusivamente il susseguirsi dei passaggi da svolgere per la risoluzione del problema ad alto livello. Sarà poi il programma ad implementare questi passaggi in qualche modo.

Paradigmi di programmazione

Per implementare un algoritmo in un programma, sarà necessario **scrivere** questo **programma**. In questo contesto, dunque, un programma è un normalissimo **testo** (il **codice sorgente**) scritto, però, in un **linguaggio di programmazione** il quale è, ovviamente, un linguaggio formale.

I numerosi linguaggi di programmazione possiedono caratteristiche diverse, ma alcune di queste sono comuni a più linguaggi e, per questo motivo, esistono dei veri e propri **paradigmi** di programmazione, tra cui:

- (ⓘ) Paradigma **imperativo**: quando si programma secondo il paradigma imperativo, si controlla direttamente il **flusso di esecuzione delle istruzioni**. Questo paradigma si suddivide a sua volta in:
  - (Ⓟ) Paradigma **procedurale**: organizzato attraverso procedure che si chiamano a vicenda.
  - (Ⓞ) Paradigma **orientato agli oggetti**: organizzato attraverso oggetti che incapsulano dati e funzionalità.

  ESEMPI: C (Ⓟ), C++ (Ⓟ+Ⓞ), Assembly (ⓘ), Lua (Ⓟ+Ⓞ), Java (Ⓞ), JavaScript (Ⓟ+Ⓞ), Python (Ⓟ+Ⓞ)
- (Ⓓ) Paradigma **dichiarativo**: quando si programma secondo il paradigma dichiarativo, si scrive il risultato che si vuole ottenere, ma non si specifica il modo in cui ottenerlo. È il computer stesso a capire quali istruzioni dovrà eseguire per ottenere il risultato che è stato descritto inizialmente. Questo paradigma si suddivide a sua volta in:
  - (Ⓕ) Paradigma **funzionale**: il risultato desiderato è dichiarato attraverso il calcolo di diverse funzioni.
  - (ⓜ) Linguaggi di **markup**: utilizzati principalmente per descrivere la struttura di documenti.

  ESEMPI: JavaScript (Ⓕ), Erlang (Ⓕ), HTML (ⓜ), LaTeX (ⓜ), SQL (Ⓓ), Prolog (Ⓓ), YAML (Ⓓ), R (Ⓕ), Python (Ⓕ)

Compilazione e interpretazione

Esistono due modi principali per eseguire programmi: **compilazione** e **interpretazione**.

Compilazione

Quando un programma viene **compilato**, il suo codice sorgente viene **tradotto in linguaggio macchina** attraverso un altro programma (il **compilatore**), creando un file **eseguibile** (.exe per Windows, .bin per Linux). Questo file, una volta eseguito, avvia il programma.

Il vantaggio è che il computer eseguirà le istruzioni **direttamente in linguaggio macchina**, un processo **estremamente veloce**. Lo svantaggio è che l'eseguibile è limitato alla sola **architettura di processore** sulla quale è stato generato.

Interpretazione

Quando un programma viene **interpretato**, il suo codice sorgente viene **letto ed eseguito istruzione per istruzione** da un altro programma (l'**interprete**).

Il vantaggio è che il codice può essere eseguito su **qualsiasi architettura** che supporti l'esecuzione dell'interprete. Lo svantaggio è che l'esecuzione del codice interpretato è generalmente **più lenta**.

Esistono pochi linguaggi che si collocano nel mezzo tra compilazione e interpretazione. Ad esempio, i programmi scritti in Java vengono compilati in un codice a basso livello detto bytecode (diverso dal linguaggio macchina) che viene poi interpretato all'interno di una Java Virtual Machine (JVM).

Linguaggi di programmazione ad alto e basso livello

I computer elaborano **sequenze di bit**, rappresentate come 0 e 1, che corrispondono a **segnali elettrici** in **circuiti microscopici**. Dunque le istruzioni devono essere **estremamente semplici**.
Tuttavia, i programmatori desiderano **creare applicazioni** senza doversi occupare di dettagli così fini. Per tale motivo, ogni linguaggio di programmazione alza il livello di astrazione rispetto alle sequenze di bit.

Alto livello (Python)

```
def function():
    return b*h/2
```

Basso livello (Assembly)

```
LOAD r1, b
LOAD r2, h
MUL r1, h
DIV r1, #2
RET
```

Linguaggio macchina

```
01001001011011011101
1010011011011011O...
```

Comprensibile dagli umani

Comprensibile dalle macchine

È proprio per questo motivo che un programma, per poter essere eseguito, ha bisogno di oltrepassare un processo di **traduzione** (tramite **compilazione** o **interpretazione**). Tale traduzione converte un qualcosa ad alto livello **in qualcosa al livello più basso**, ovvero in linguaggio macchina.

Fortunatamente, questo processo viene svolto in **automatico** da programmi appositi.

Cos’è un computer (tecnicamente)

Digitalizzazione delle informazioni

Linguaggi di programmazione

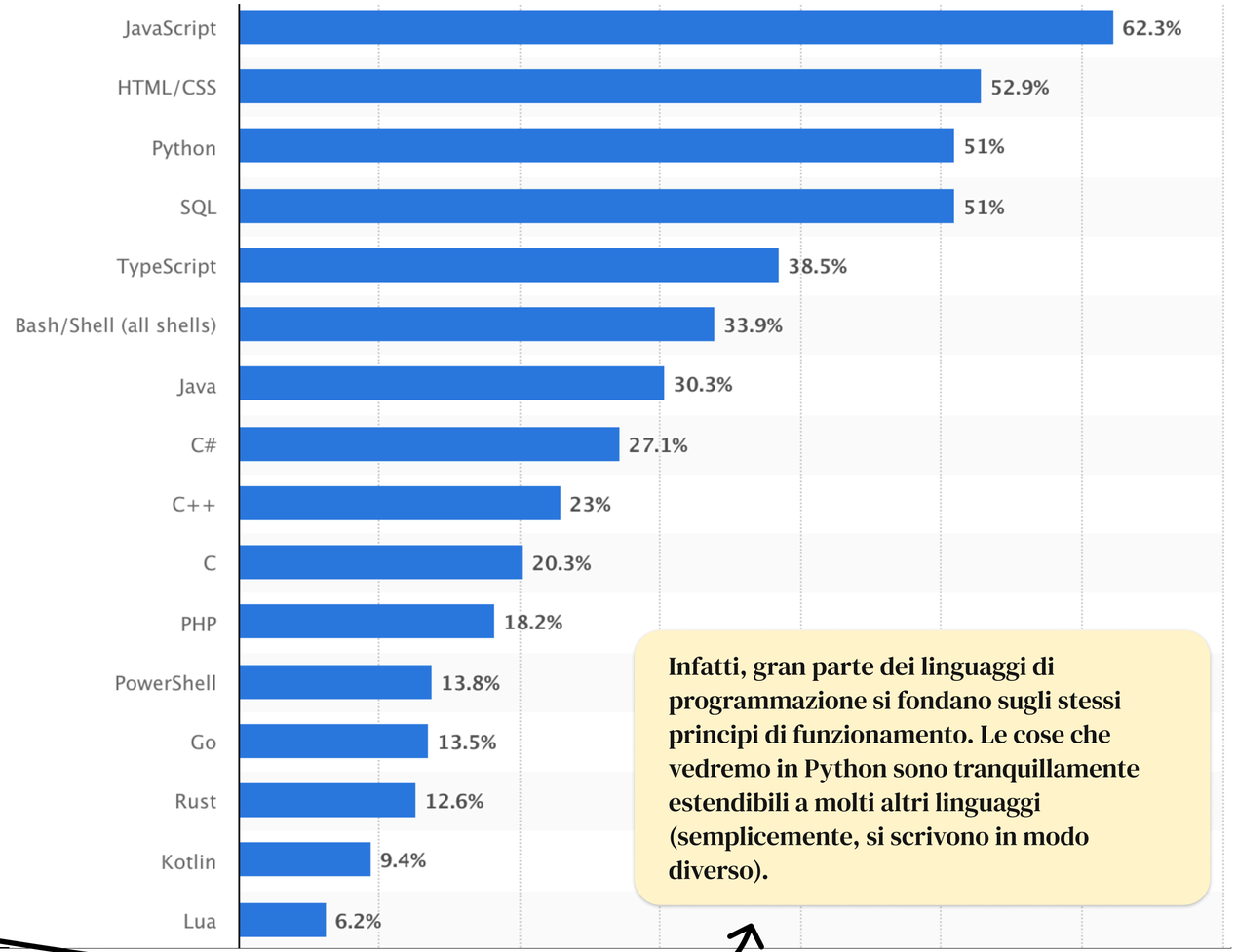
→ **Conclusioni**

Perché Python?

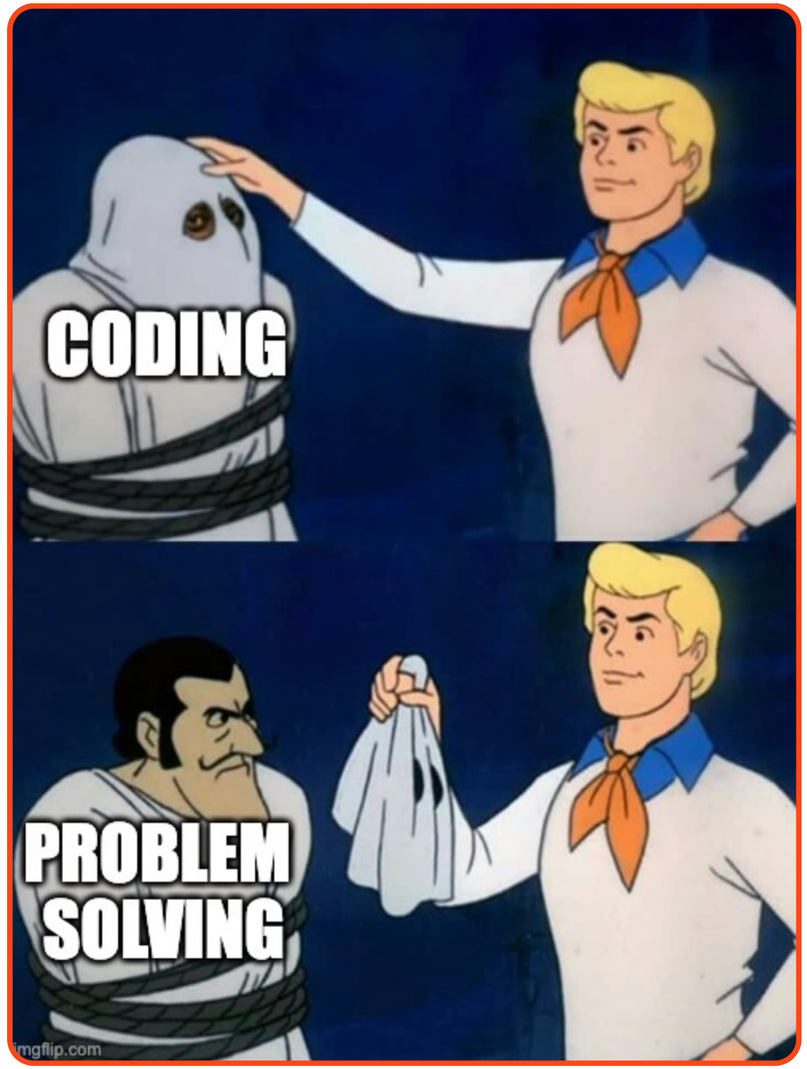
Beh, ovviamente perché me lo avete chiesto.

Però, oltre a questo, **Python** è un'ottima scelta per iniziare grazie alla sua **sintassi semplice**, alle numerosissime applicazioni e, soprattutto, al suo ampio utilizzo.

Ritengo che una scelta ancora migliore sia **C** o **C++**, ma entrambi, essendo **più a basso livello**, necessitano di una **comprensione molto più approfondita** dei concetti di base. In ogni caso, nonostante i linguaggi siano spesso diversi tra loro, impararne uno **efficacemente** semplifica il processo di apprendimento di tutti gli altri.



Infatti, gran parte dei linguaggi di programmazione si fondano sugli stessi principi di funzionamento. Le cose che vedremo in Python sono tranquillamente estendibili a molti altri linguaggi (semplicemente, si scrivono in modo diverso).



Perché programmare?

Anche in questo caso, ovviamente, è perché me lo avete chiesto.
Però, ottima scelta. Saper programmare è una **skill importante** e molto **utile**.

Certamente si impara a programmare per **creare applicazioni** o **automatizzare processi**, ma anche (e soprattutto) perché imparando a programmare si acquisisce una **mentalità rigorosa** e si impara a **risolvere problemi**.

Voi dopo che avrete imparato a programmare

Grazieng!

DALLA PROSSIMA COMINCIAMO A PROGRAMMARE – INTANTO DATEMI QUALSIASI TIPO DI FEEDBACK PER MIGLIORARE QUALUNQUE COSA