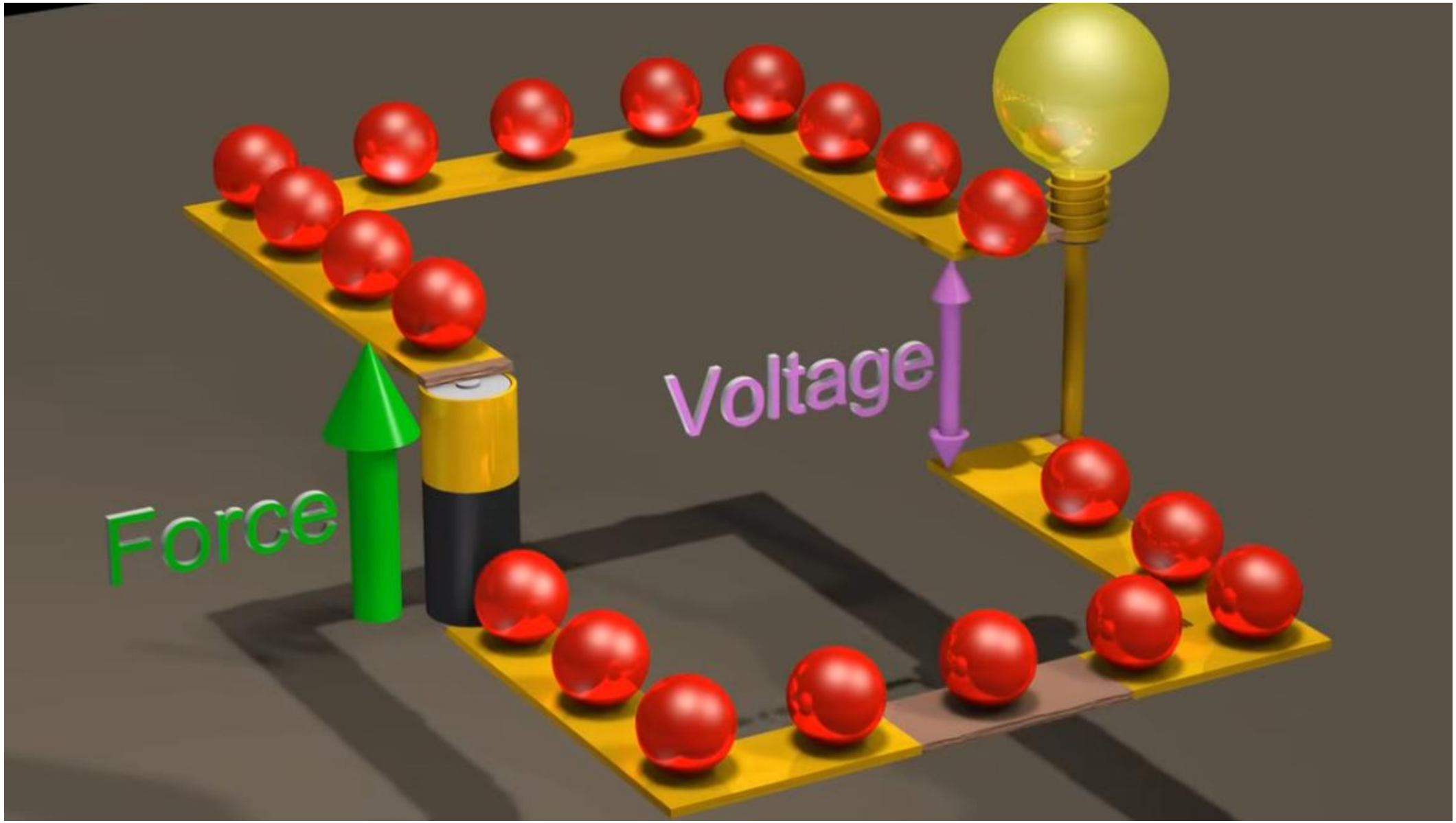
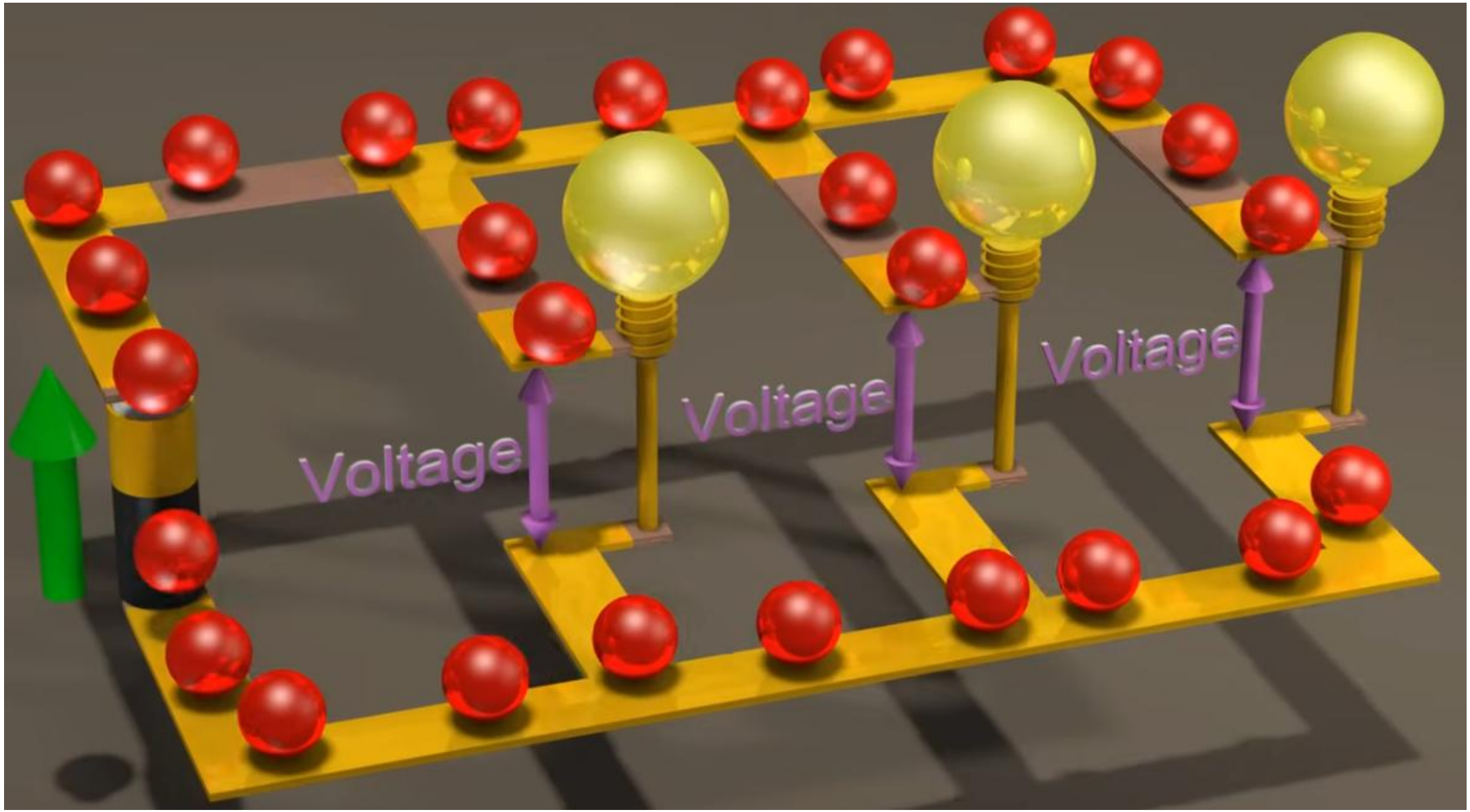
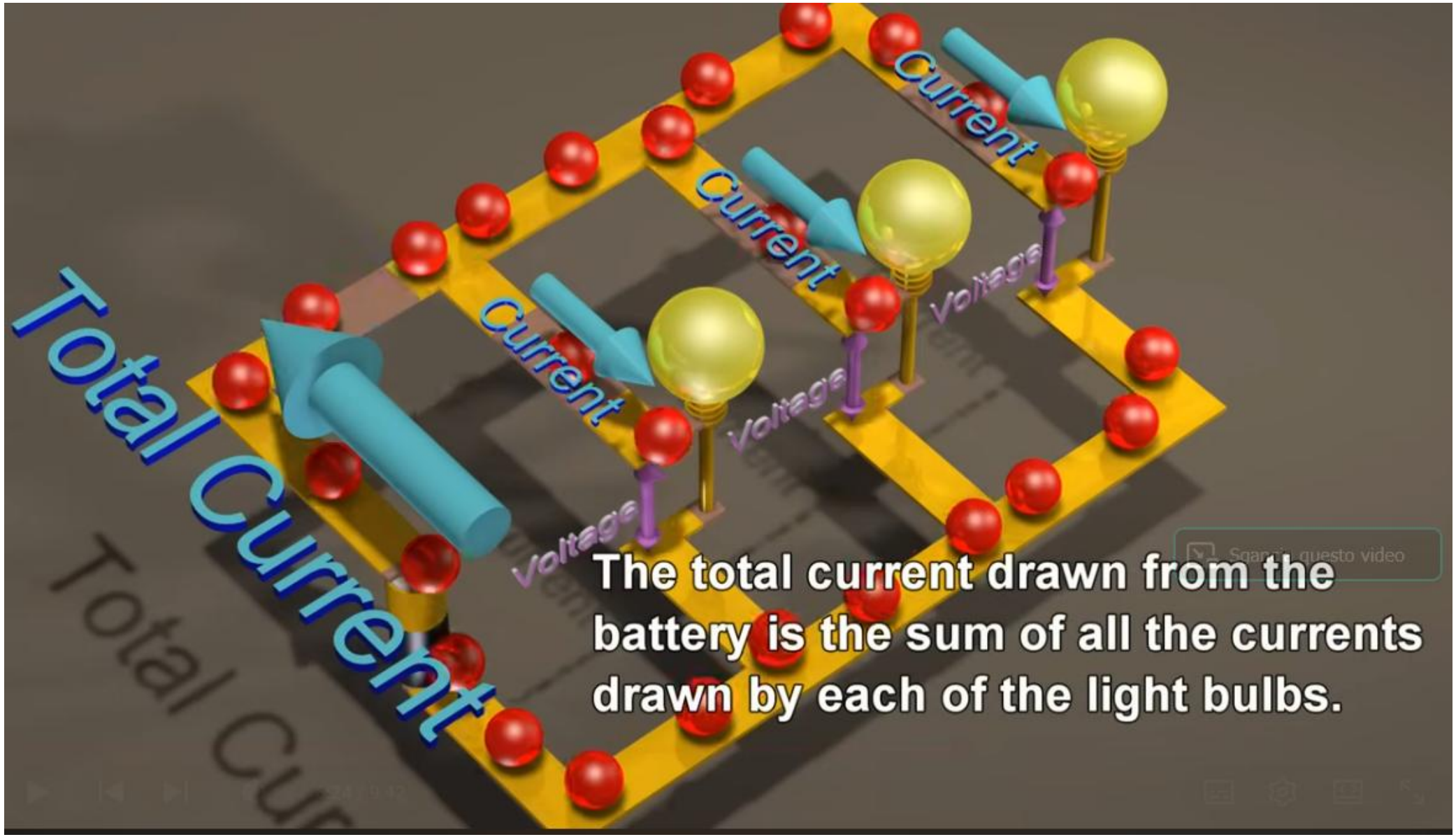
# CORSO AVANZATO DI INFORMATICA E ROBOTICA

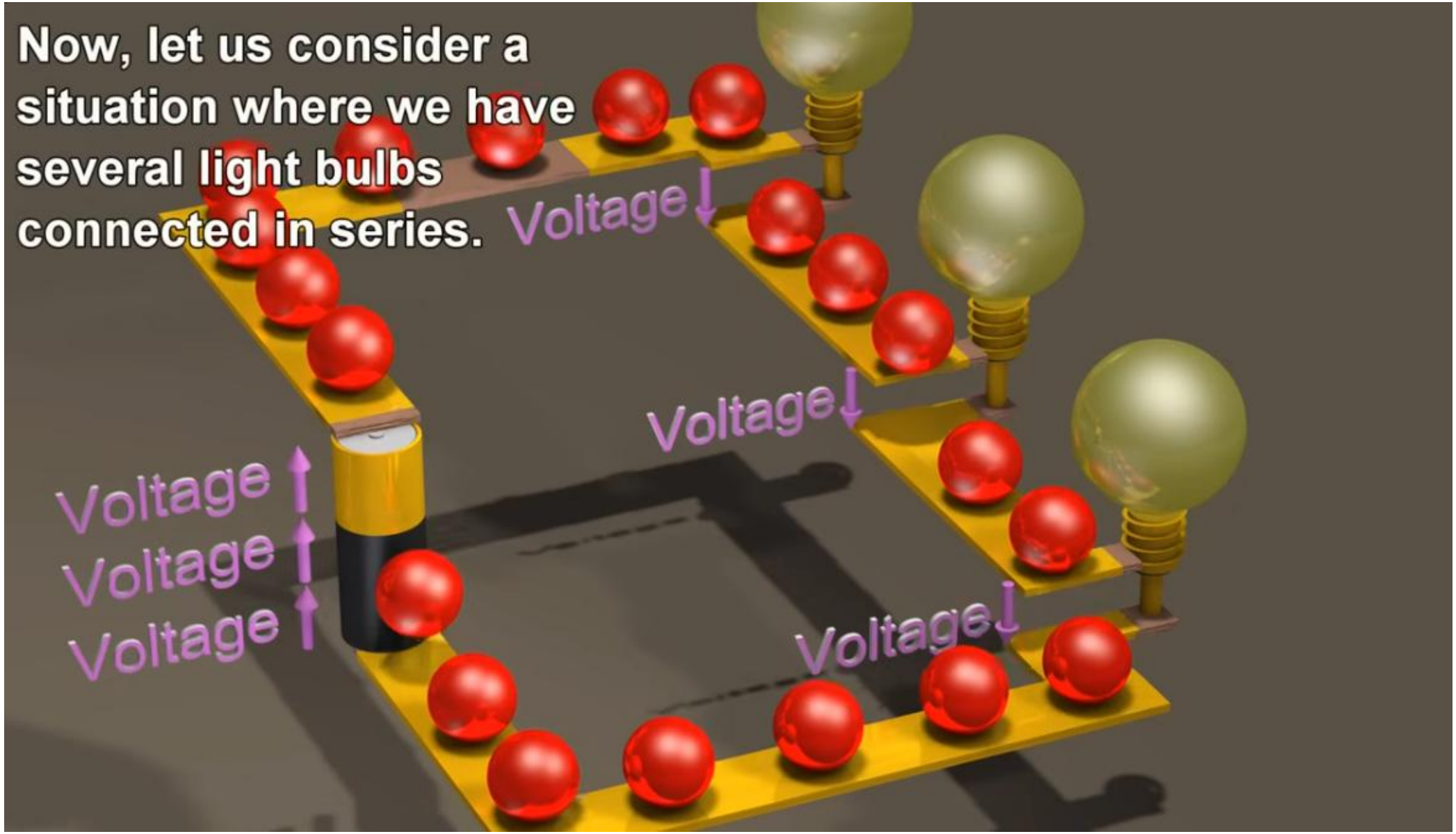
## LEZIONE 2: ARDUINO

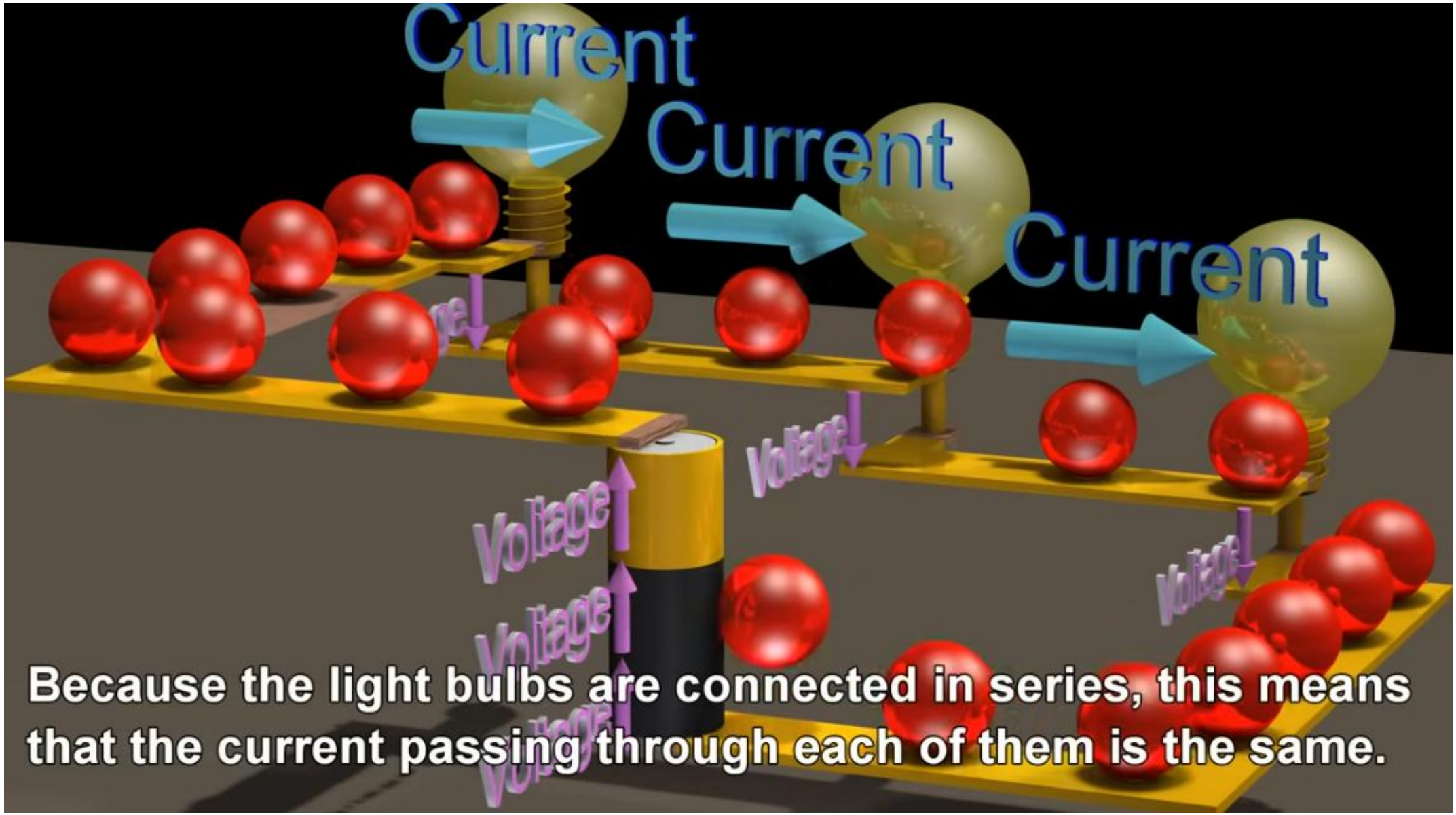
Carlo Zambaldo | 12 novembre 2025


Force
Voltage


Voltage
Voltage
Voltage


Total Current
Current
Current
Current
Voltage
Voltage
Voltage
The total current drawn from the battery is the sum of all the currents drawn by each of the light bulbs.


Now, let us consider a situation where we have several light bulbs connected in series.
Voltage
Voltage
Voltage
Voltage
Voltage
Voltage


Current
Current
Current
Voltage
Voltage
Voltage
Voltage
Because the light bulbs are connected in series, this means that the current passing through each of them is the same.

# Piattaforme di Prototipazione

- Arduino è una piattaforma basata su microcontrollore, ideale per progetti che richiedono il controllo di sensori e motori,
- Raspberry Pi è un mini-computer più potente, adatto a progetti complessi che necessitano di un sistema operativo e connessioni di rete.
- La scelta tra le due dipende dall'applicazione: Arduino per il controllo in tempo reale e Raspberry Pi per compiti che richiedono maggiore potenza di calcolo, come un server.

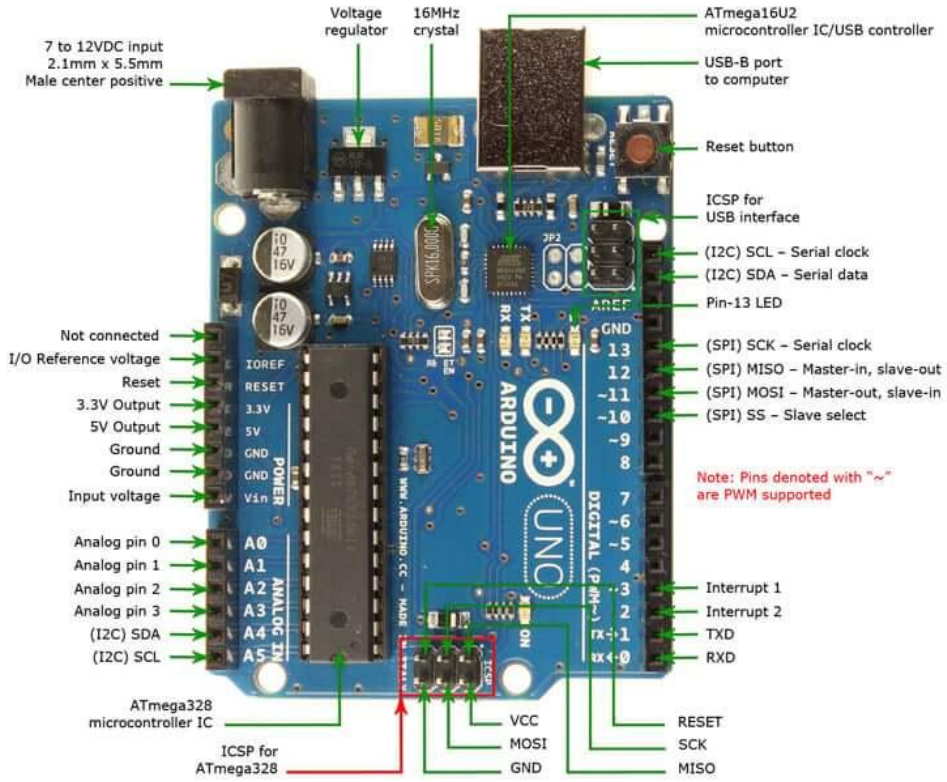

# Architettura di un Microcontrollore (MCU) - Arduino

Il cuore di Arduino UNO è il microcontrollore ATmega328P, un sistema completo su un singolo chip.

- CPU: Esegue le istruzioni dello sketch.
- Memoria Flash: Memoria non volatile dove risiede il programma.
- SRAM: Memoria volatile per le variabili usate a runtime.
- EEPROM: Memoria non volatile per dati da conservare allo spegnimento.

- Periferiche Integrate:
  - Convertitore A/D (ADC): per leggere segnali analogici.
  - Timers/Counters: Per generare segnali PWM e gestire interrupt temporizzati.
  - Interfacce di Comunicazione: UART (seriale), SPI, I²C.

- Non ha un sistema operativo, ma esegue un singolo programma (sketch) alla volta.

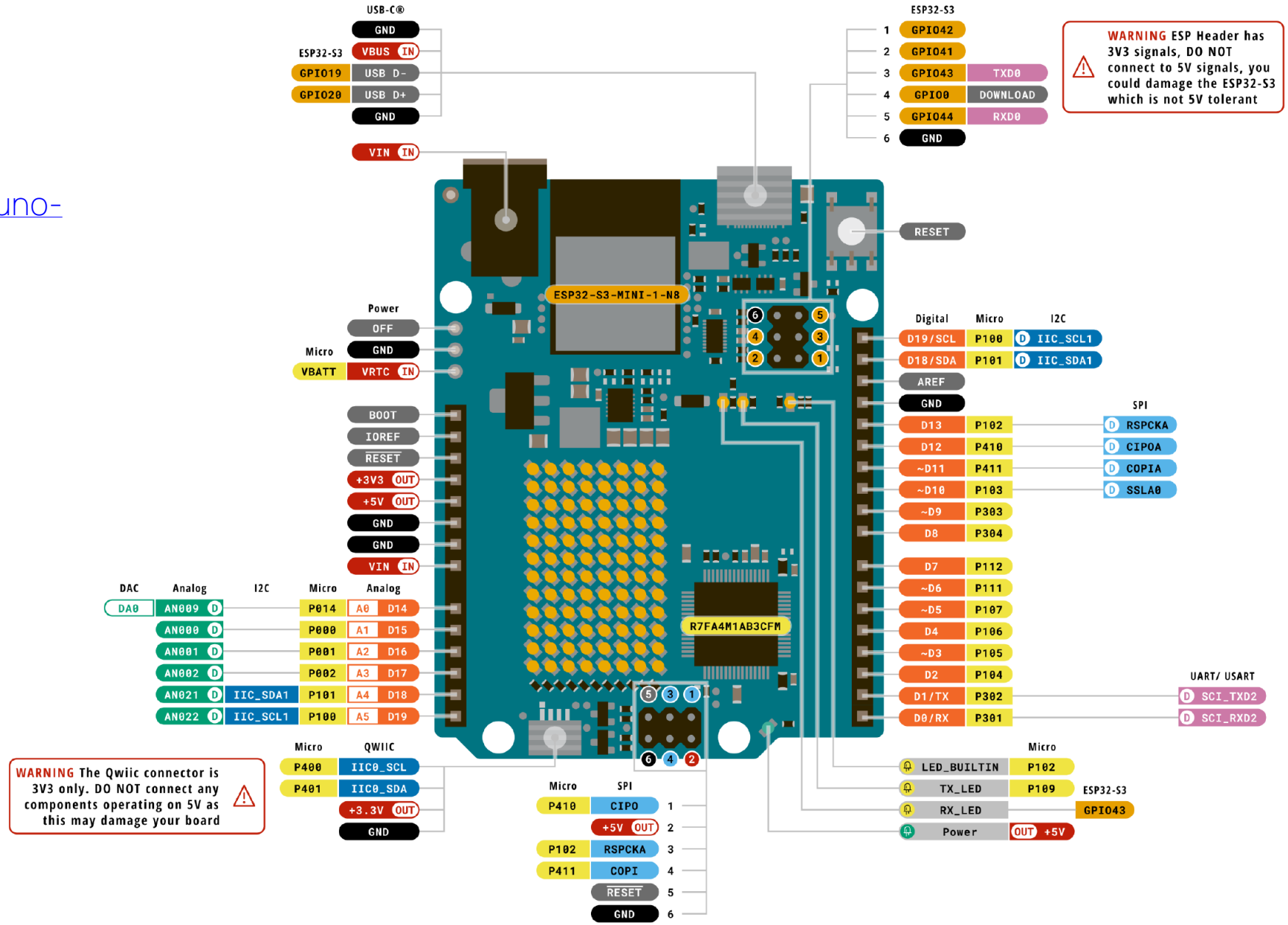


# Architettura di un Single-Board Computer (SBC) - Raspberry Pi

- Raspberry Pi è un mini-computer completo basato su un System-on-Chip (SoC).
- SoC (es. Broadcom BCM2711): Integra più componenti in un unico chip:
  - CPU (Multi-core ARM): Processore ad alte prestazioni per eseguire un sistema operativo.
  - GPU (VideoCore): Unità di elaborazione grafica.
  - Controller I/O: Gestisce USB, Ethernet, GPIO.
- RAM: Memoria di sistema esterna al SoC
- Sistema Operativo: Esegue un OS completo (es. Raspberry Pi OS, una derivata di Debian Linux), che gestisce i processi, la memoria e le periferiche.

# Arduino UNO R4 WiFi

Cheat-sheet:
https://docs.arduino.cc/tutorials/uno-r4-wifi/cheat-sheet/



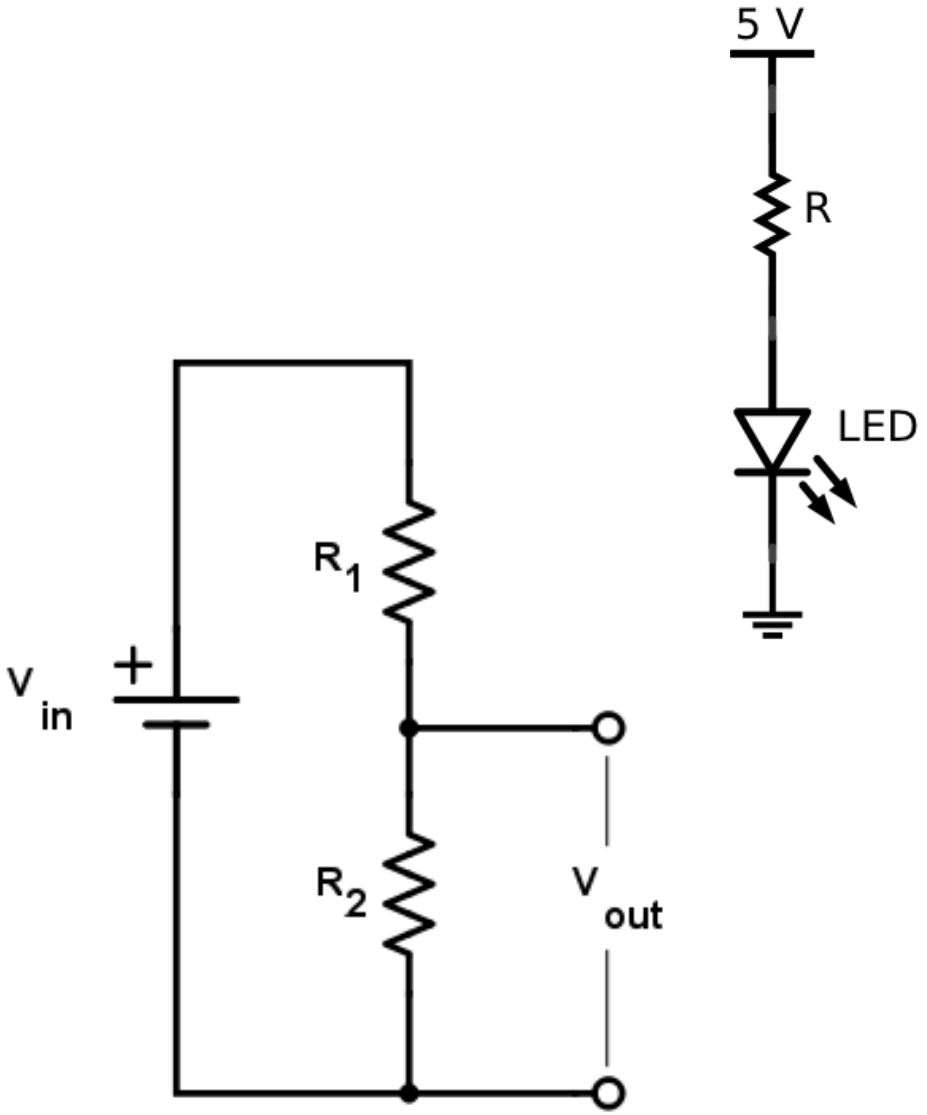
# ARDUINO

I pin GPIO sono l'interfaccia fisica tra il controller e il mondo esterno.
Modalità del Pin:

- INPUT: Alta impedenza, legge lo stato logico esterno.
- OUTPUT: Bassa impedenza, impone uno stato logico (HIGH/LOW).
- INPUT_PULLUP: Attiva una resistenza interna verso VCC, utile per leggere pulsanti senza resistore esterno.

Caratteristiche Elettriche:

- Livelli Logici: Le tensioni che definiscono HIGH e LOW (es. TTL, CMOS). Su Arduino 5V, su Raspberry Pi 3.3V. Attenzione: un segnale a 5V può danneggiare un pin di Raspberry Pi!
- Corrente Massima (Source/Sink): Ogni pin può erogare/assorbire una corrente limitata (es. ~20-40 mA per ATmega328P). Superarla causa danni permanenti.



# Segnali Digitali vs. Analogici

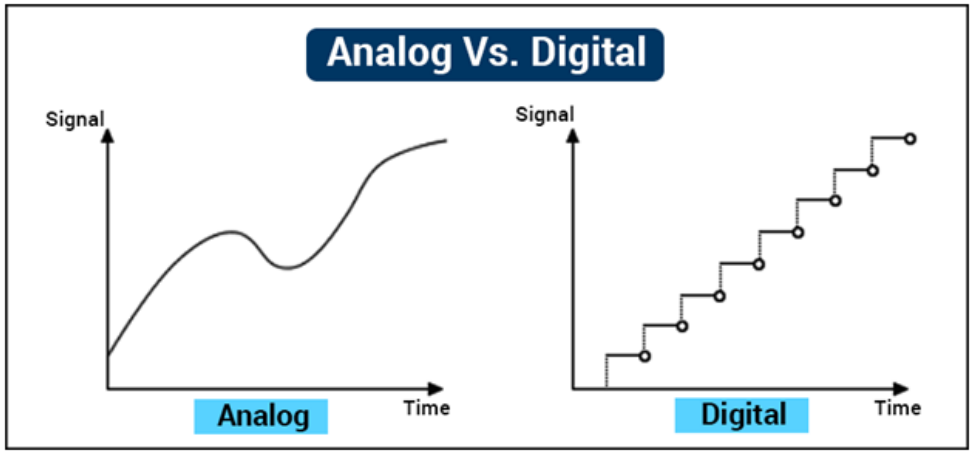
Segnale Digitale:

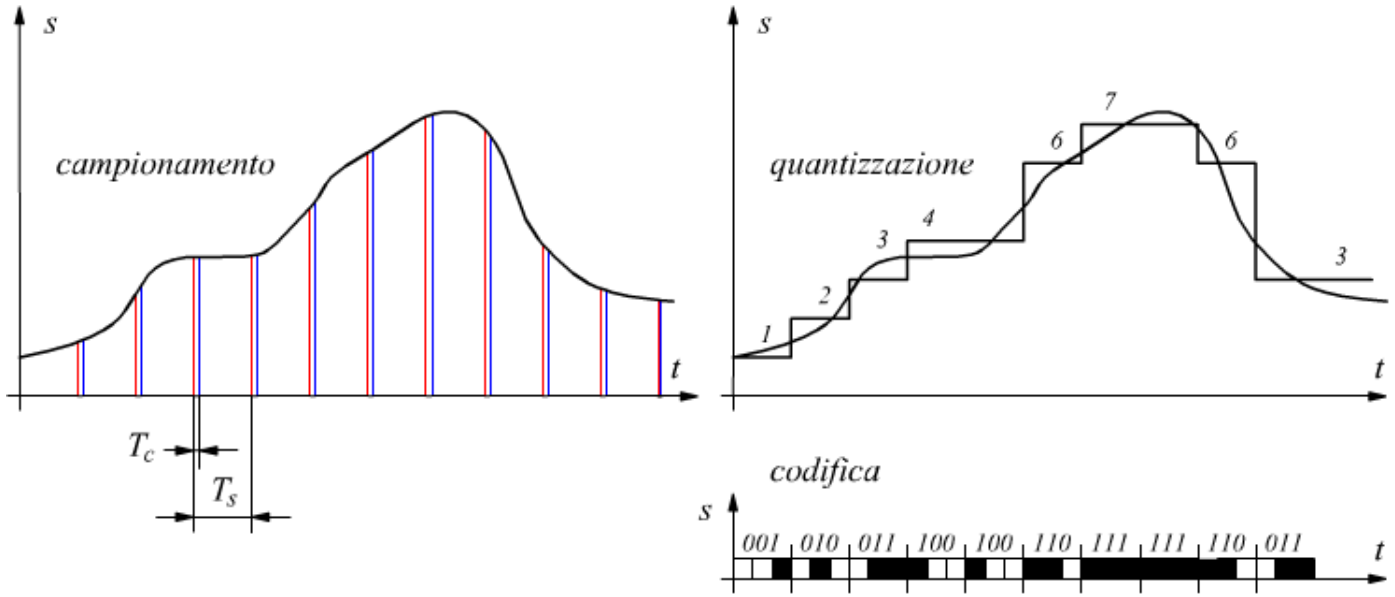
- Discreto nel valore e nel tempo.
- Rappresentato da due livelli di tensione: HIGH (es. > 3V) e LOW (es. < 0.8V).
- Utilizzato per comunicare stati binari (ON/OFF, 1/0).

Segnale Analogico:

- Continuo nel valore e nel tempo.
- La tensione può assumere qualsiasi valore in un range definito (es. da 0V a 5V).
- Necessario per misurare grandezze fisiche continue (temperatura, luminosità).

- Vedi convertitori A/D e D/A



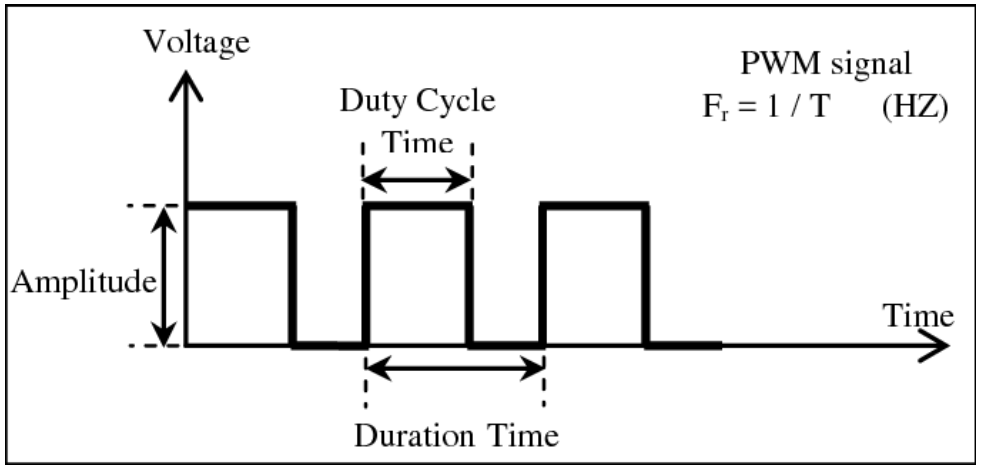
# Da Analogico a Digitale e Viceversa

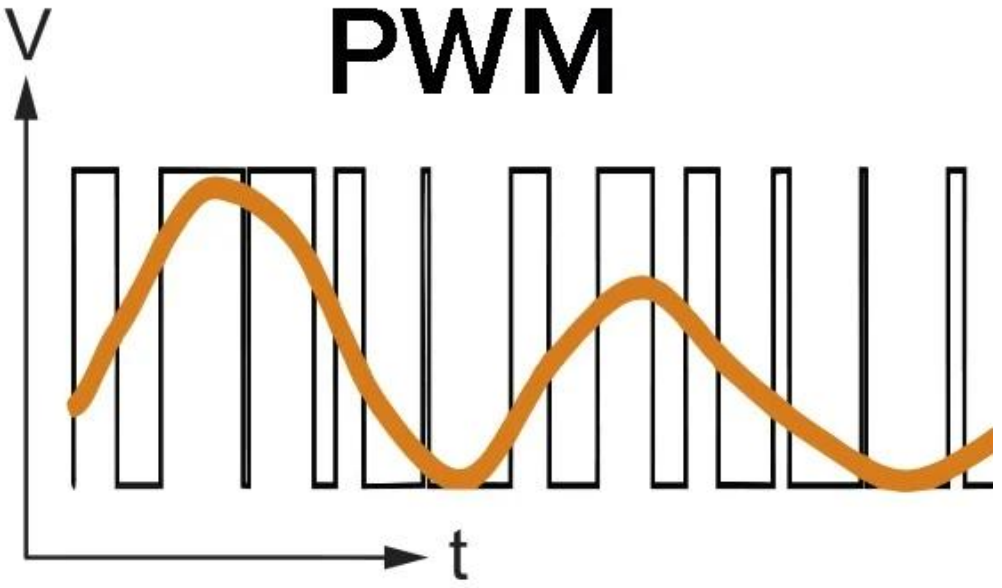
ADC (Analog-to-Digital Converter):

- Converte una tensione analogica in un valore numerico.
- Risoluzione: Il numero di bit usati per la conversione. L'ADC di Arduino ha (ad esempio) 10-bit, quindi può rappresentare 210=1024 livelli discreti (da 0 a 1023).

PWM (Pulse Width Modulation):

- Una tecnica per simulare un'uscita analogica usando un pin digitale.
- Si basa sulla variazione del Duty Cycle di un'onda quadra a frequenza fissa.
- Duty Cycle: La percentuale di tempo in cui il segnale rimane HIGH all'interno di un periodo. Un duty cycle del 100% equivale a VCC, del 50% a VCC/2, dello 0% a GND



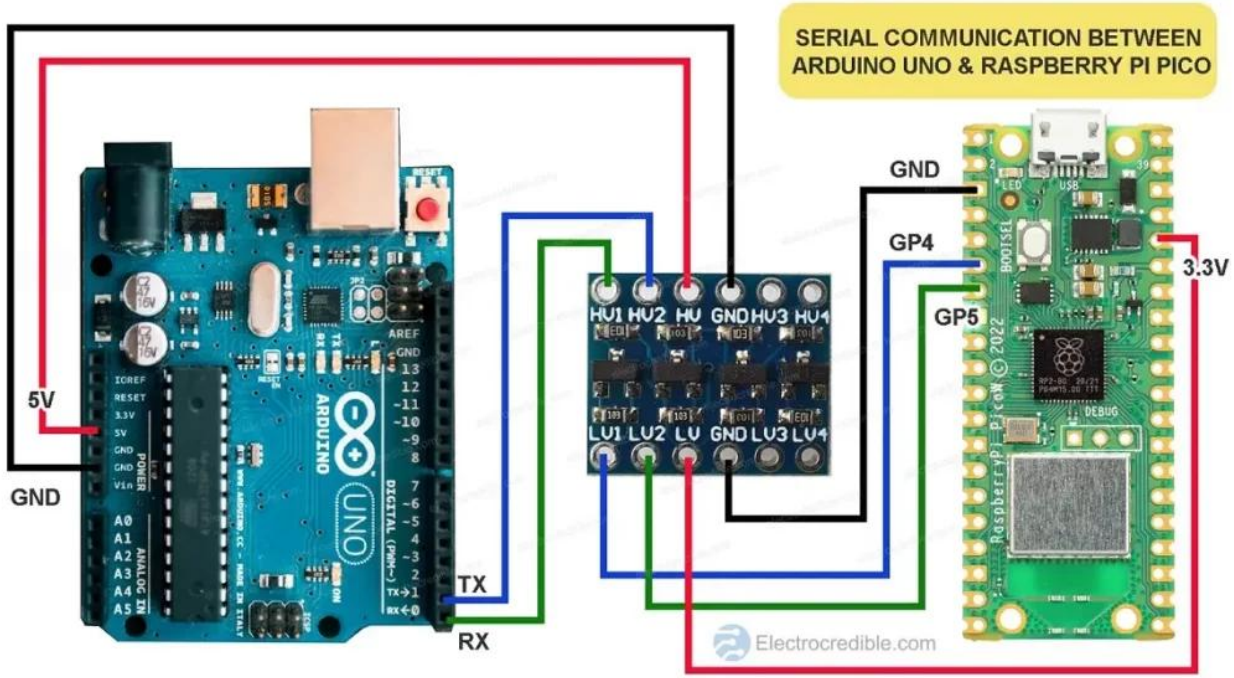
# Codifica dell'informazione

I microcontrollori non capiscono "acceso" o "spento", capiscono solo livelli di tensione.
Arduino UNO (logica a 5V):

- LOW: Una tensione vicina a 0V (<2.5 V).
- HIGH: Una tensione vicina a 5V (>2.5 V).

Raspberry Pi (logica a 3.3V):

- LOW: Una tensione vicina a 0V.
- HIGH: Una tensione vicina a 3.3V.




Attenzione: Questa differenza non è un dettaglio, è una regola critica. Se un pin di Arduino in modalità OUTPUT (che eroga 5V) viene collegato a un pin GPIO di un Raspberry Pi, lo danneggerà permanentemente.
I pin del Raspberry Pi non sono "tolleranti" ai 5V. Per far comunicare questi due dispositivi, servono dei circuiti appositi chiamati level shifter (convertitori di livello logico).

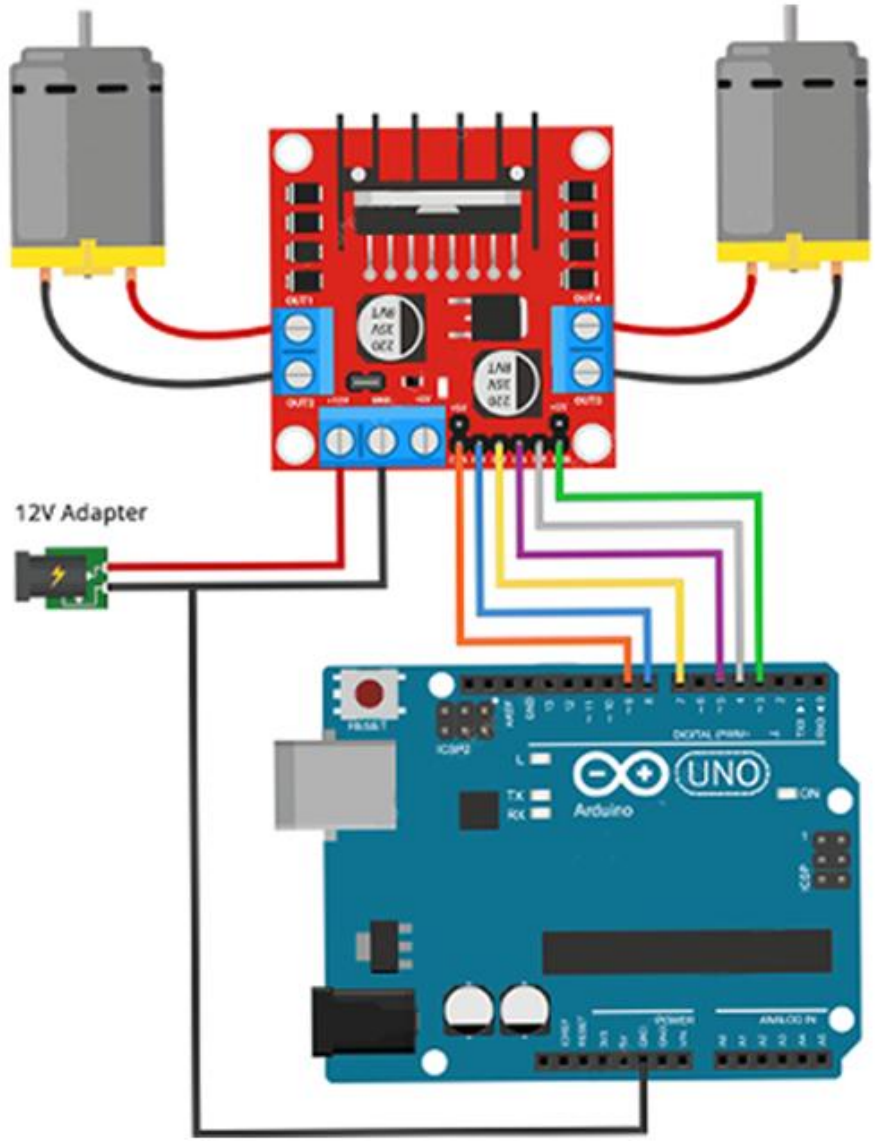
# Interfacce di Input e Output

Sensori (Input):

- Analogici: Forniscono una tensione continua (es. fotoresistenza, sensore di temperatura LM35). Si leggono con l'ADC.
- Digitali: Forniscono uno stato HIGH/LOW (es. pulsante, sensore PIR).
- Basati su Protocollo: Comunicano tramite protocolli seriali standard (es. sensore di umidità DHT11, sensori I²C/SPI).

Attuatori (Output):

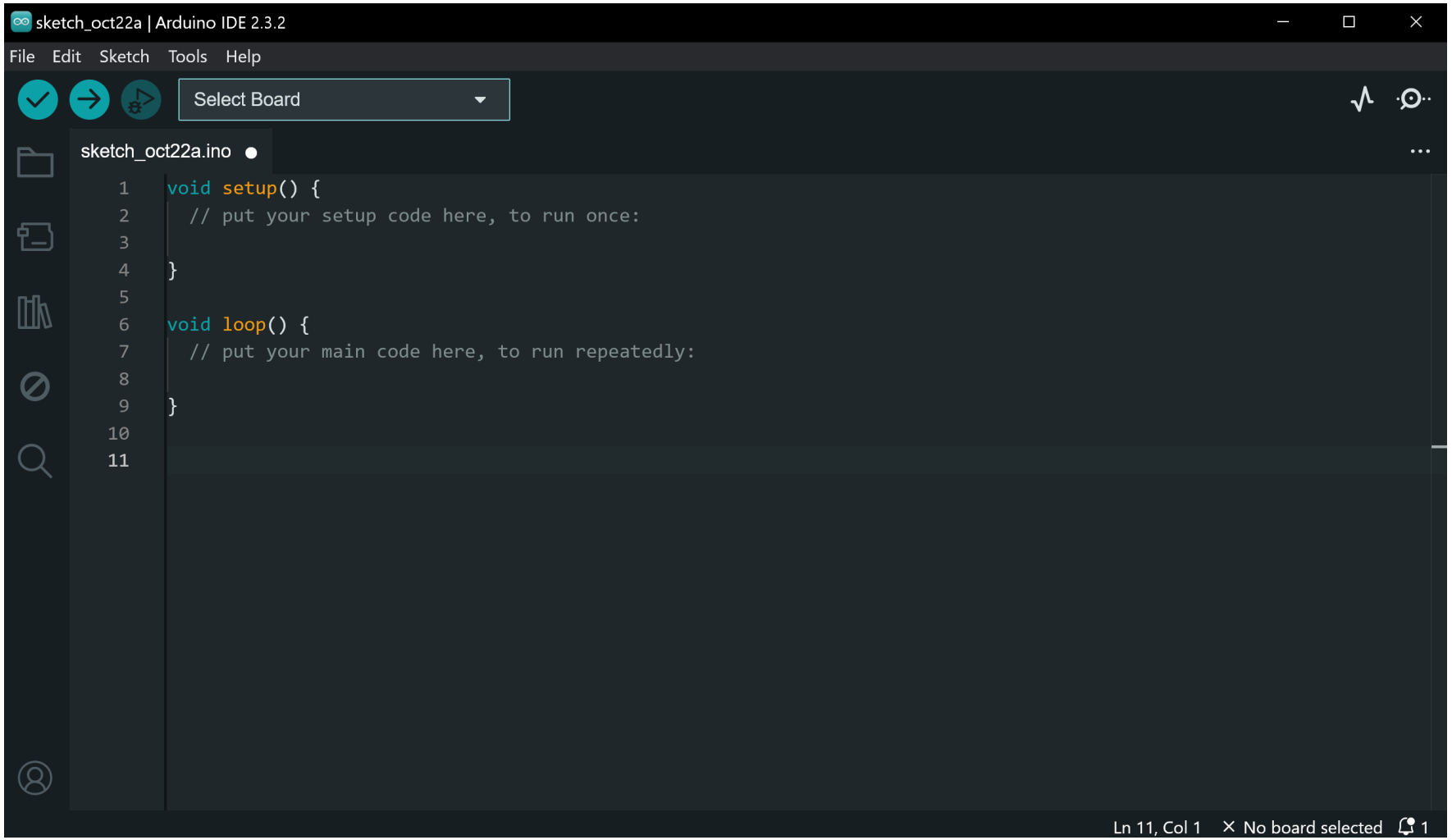
- Semplici: Controllati direttamente da un pin GPIO (con resistenza di limitazione) (es. LED).
- Complessi: Richiedono circuiti di pilotaggio esterni (driver) per gestire correnti elevate (es. Motori DC, servomotori, relay).



# Codice Arduino

1. Installare Arduino IDE da: https://www.arduino.cc/en/software/
2. Aprire Arduino IDE
3. Selezionare la Board

Se non avete ancora Arduino potente sempre “smanettare” su Tinkercad: https://www.tinkercad.com/



# Review generale del codice C++

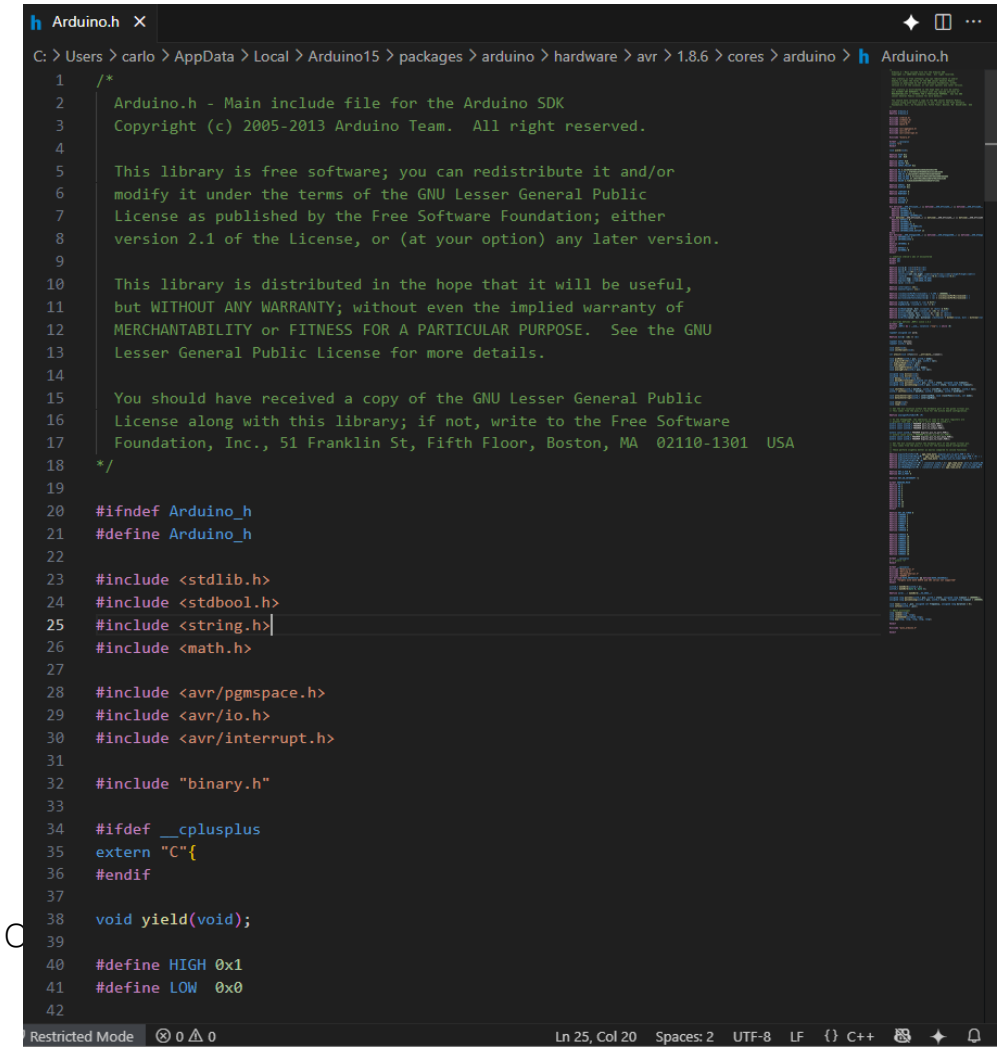
- #define – Direttiva del preprocessore: #define <nome> <valore>
- const int – Costante di tipo: const int LED_RED = 13;
- Functions:

```
<return type> <function name>(<input args>)}
        // code to execute
    }
```

- Arrays

```
int primes[5] = {2,3,5,7,11}
int number = primes[0];
```

- Arduino ha la Comunicazione Seriale:
  - Porta seriale
  - Serial.begin(9600); // baudrate determina la velocità di trasmissio
  - Serial.print() oppure Serial.println()
  - Esiste anche la controparte per leggere da porta seriale




https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/Ardui

# I/O pins

- Digital Pins (0–13)
  - Leggere tensioni LOW (<2.5V) e HIGH (>2.5V or 5V)
  - Per leggere usare:
    digitalRead(<pin number>)
  - Per scrivere usare:
    digitalWrite(<pin number>, <HIGH or LOW>)
- PWM Pins (~3, 5, 6, 9, 10, 11)
  - Simulano un segnale analogico tramite Pulse Width Modulation
  - analogWrite(<pin number>, <value>)
    dove <value> varia da 0 (a 0 V) a 255 (a 5 V)
- Analog Pins (A0–A5)
  - Misurano tensioni analogiche tra 0 e 5 V
  - Restituiscono un valore numerico tra 0 e 1023
    analogRead(<pin number>)

```
#define BUTTON_PIN 2
int variabileGlobale = 1;

void setup(){
    Serial.begin(9600);
    pinMode(BUTTON_PIN,INPUT);
    pinMode(8,OUTPUT);
}

void loop(){
    int buttonState = digitalRead(BUTTON_PIN);
    Serial.println(buttonState);
}
```

Tutti i pin possono essere configurati come INPUT o OUTPUT con
pinMode(<pin number>, INPUT or OUTPUT)

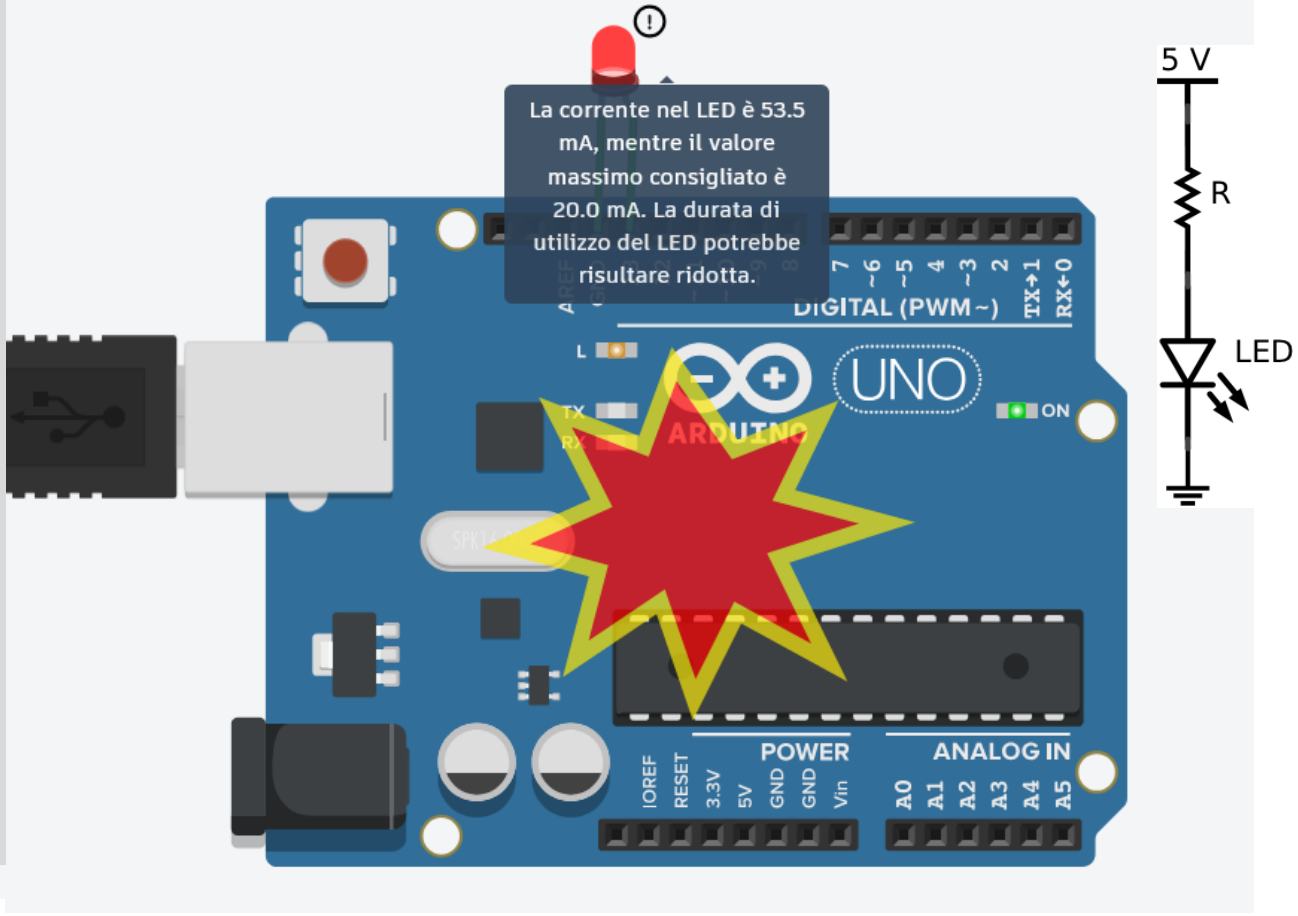
# Esercizio 1 – Blink

```
#define LED_PIN 13

void setup(){
    pinMode(LED_PIN,OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

Metteteci un resistore da 220 Ohm!!



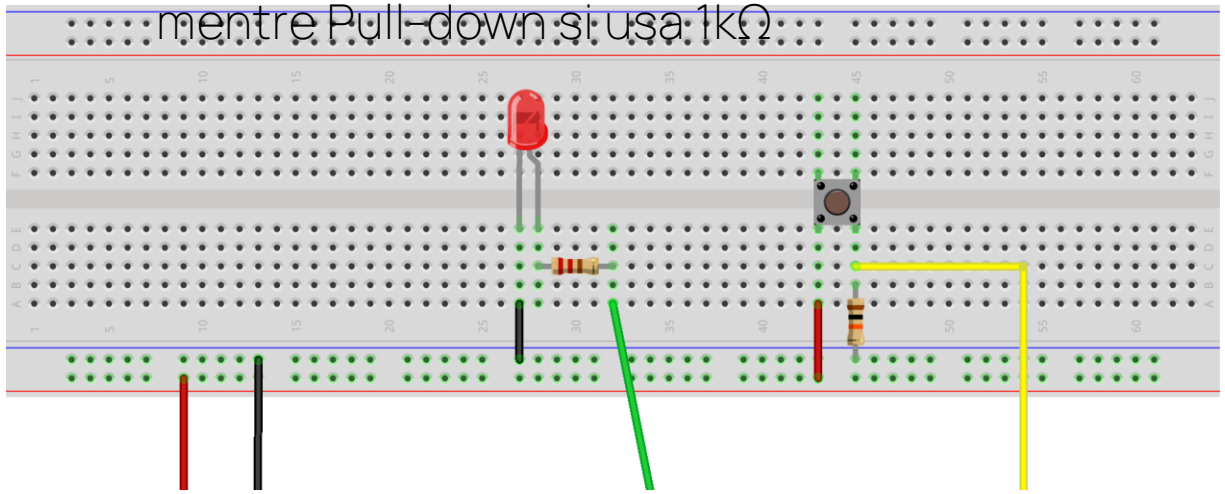
# Esercizio 2 – Aggiungere un Pulsante

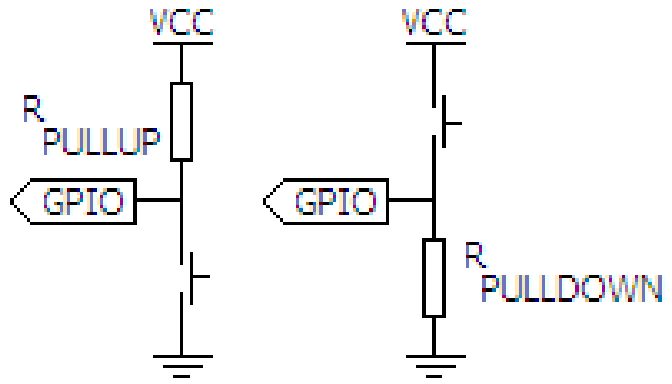
```
#define LED_PIN 13
#define BUTTON_PIN 2

void setup(){
     pinMode(LED_PIN,OUTPUT);
     pinMode(BUTTON_PIN,INPUT);
}

void loop() {
     int buttonState = digitalRead(BUTTON_PIN);
     if (buttonState == HIGH) {
               digitalWrite(LED_PIN, HIGH);
     } else {
               digitalWrite(LED_PIN, LOW);
     }
}
```

Nota: generalmente il resistore per i LED è 220Ω,
mentre Pull-down si usa 1kΩ



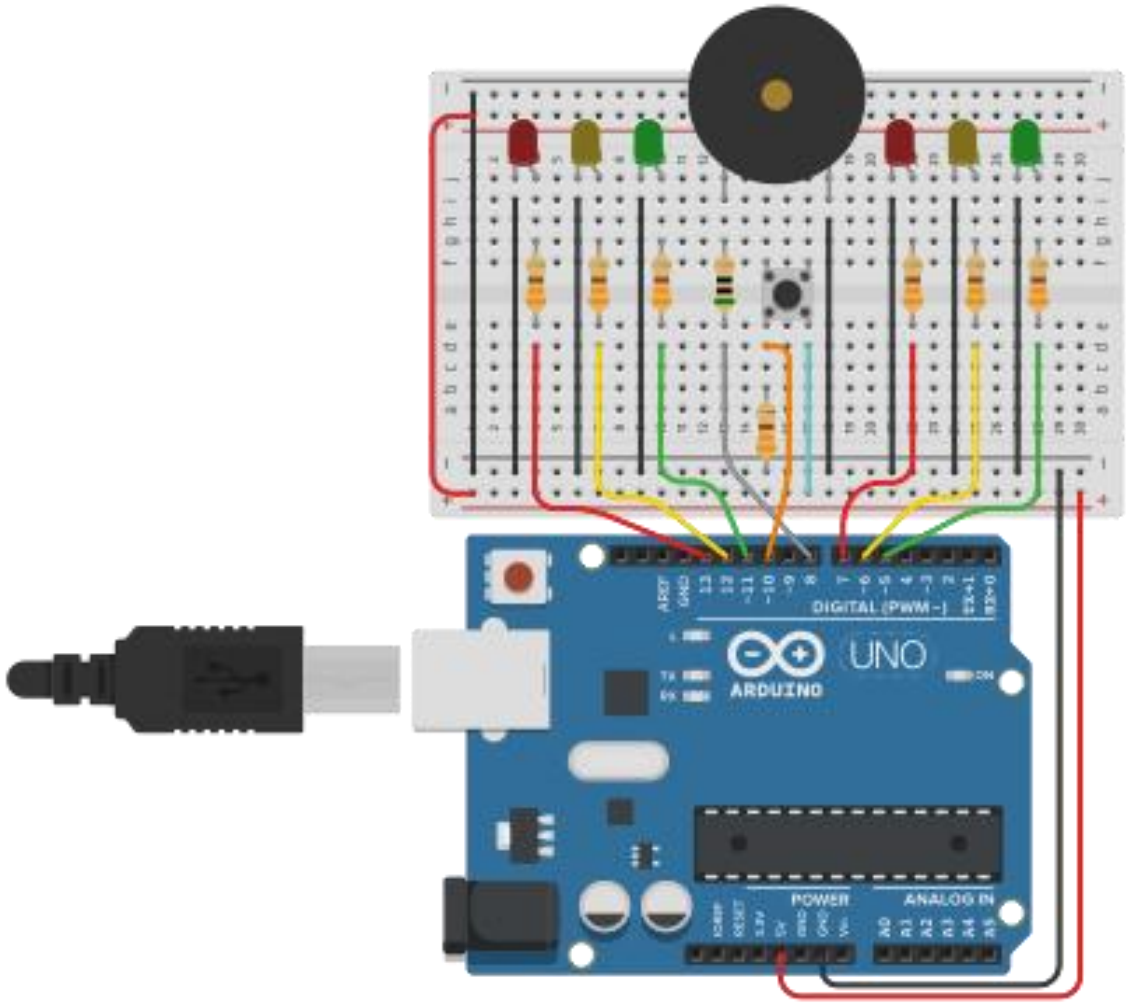



https://sistemiereti.com/resistenza-pull-up-pull-down/

# Esercizio 3 – Semaforo controllato da pulsante di richiesta e segnale sonoro

Due semafori (uno per i pedoni e uno per le macchine) che simuli una situazione di vita reale in cui sono presenti due semafori che indicano il passaggio delle macchine o dei pedoni.

Nel circuito è stato inserito un pulsante di richiesta per il transito dei pedoni con un segnale sonoro che indica la possibilità o meno di attraversamento della strada da parte delle persone non vedenti.

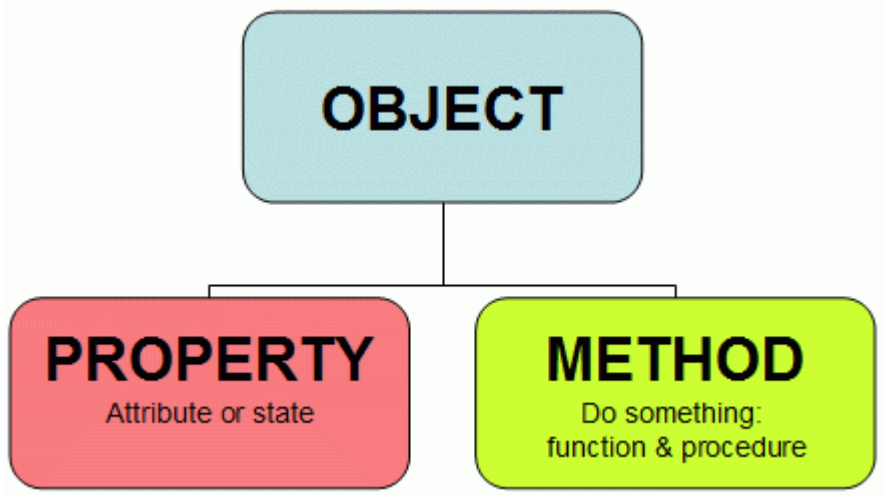


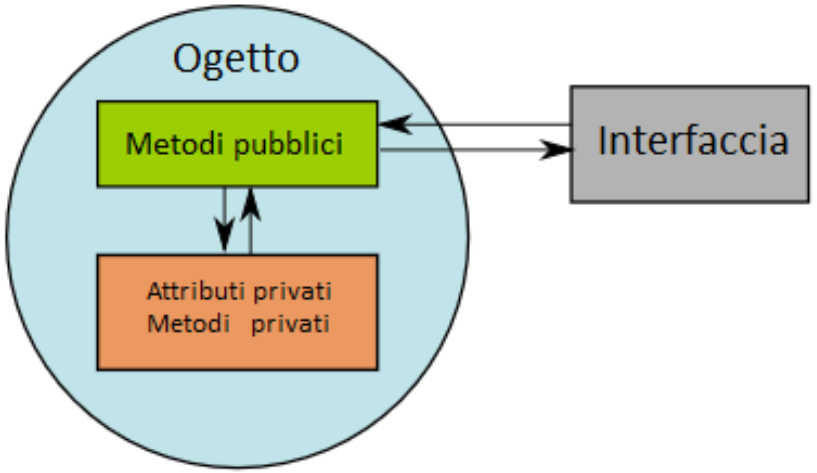

https://www.tinkercad.com/things/k3K7ANAebpL-semaforo-con-pulsante-di-richiesta-e-segnale-sonoro-su-arduino

# Programmazione Orientata agli Oggetti (OOP)

- Le classi e l'OOP (Object-Oriented Programming) sono un modo di organizzare il codice per renderlo più chiaro, modulare e riutilizzabile.

Una classe è come un *modello* o *stampo* (una "ricetta") per creare oggetti.
Definisce:
- variabili → chiamate *attributi* (proprietà dell'oggetto)
- funzioni → chiamate *metodi* (azioni che l'oggetto può fare)



# Programmazione Orientata agli Oggetti (OOP)

«definisco» la strutturae il tipo dell'oggetto

Una classe è come un *modello* o *stampo* (una "ricetta") per creare oggetti.
Definisce:

- variabili → chiamate *attributi* (proprietà dell'oggetto)
- funzioni → chiamate *metodi* (azioni che l'oggetto può fare)

```
class LED {…}
```

«uso» la classe (oggetto reale in esecuzione)

Un'istanza è un oggetto concreto creato a partire da una classe.
È la "copia reale" costruita con la ricetta della classe.

```
LED mioled(12);
```

# Concetti base dell'OOP

| Concetto | Significato | Esempio |
|---|---|---|
| Classe | Modello | "Ricetta per un tipo di oggetto" |
| Oggetto | Istanza della classe | LED rosso(13) crea un LED reale nel programma |
| Attributi | Dati dell'oggetto | Il numero di pin, lo stato acceso/spento |
| Metodi | Azioni che l'oggetto sa fare | accendi(), spegni() |
| Costruttore | Funzione speciale che inizializza l'oggetto | imposta pinMode() |
| Incapsulamento | Ogni oggetto gestisce da solo i propri dati | non serve accedere alle variabili interne |
| Ereditarietà | Una classe può estendere un'altra | es. Semaforo può derivare da 3 oggetti LED |
| Polimorfismo | Oggetti diversi possono avere lo stesso metodo ma comportamenti diversi | muovi() può significare "ruota ruota" per un motore o "apri" per un braccio robotico |

# OOP: perché è utile nella robotica

- Ogni sensore, motore o modulo può diventare un oggetto.
- Potete creare tanti robot diversi riutilizzando le stesse classi.
- Il codice diventa più leggibile e mantenibile.
- È il modo standard in cui vengono scritte le librerie Arduino (e non solo).

- Anziché pensare in termini di funzioni isolate, pensate in termini di oggetti che rappresentano cose reali e interagiscono tra loro.

| Senza OOP | Con OOP |
|---|---|
| Hai tante funzioni sparse (moveMotor(), readSensor(), ecc.) | Hai oggetti come Motore e Sensore che sanno già come comportarsi |
| Difficile gestire tanti componenti diversi | Ogni oggetto ha il suo “comportamento” definito dentro la classe |
| Codice lungo e confuso | Codice più corto, modulare e facile da estendere |

# OOP: un esempio applicato a LED e Arduino

```
class LED {
    public:
            int pin;                    // attributo: il numero di pin
            LED(int p) {                // costruttore: si esegue quando creo l'oggetto
                    pin = p;
                    pinMode(pin, OUTPUT);
            }
            void accendi() {
                    digitalWrite(pin, HIGH);
            }
            void spegni() {
                    digitalWrite(pin, LOW);
            }
};
```

Nota: «public» significa che tutto quello che segue (fino a quando non scrivi «private» o «protected» ) è visibile all'esterno della classe

# OOP: un esempio applicato a LED e Arduino

```
LED rosso(13);
LED verde(12);

void setup() {
    // I costruttori impostano già i pin come OUTPUT <<<<<< !!!
}
void loop() {
    rosso.accendi();
    delay(1000);
    rosso.spegni();
    verde.accendi();
    delay(1000);
    verde.spegni();
}
```

# OOP: public, protected and private modifiers

| Modificatore | Visibile nella classe | Visibile nelle classi derivate | Visibile dall'esterno |
|---|---|---|---|
| `public` | ✅ | ✅ | ✅ |
| `protected` | ✅ | ✅ | ❌ |
| `private` | ✅ | ❌ | ❌ |

protected

- Si usa quando vuoi nascondere un attributo al mondo esterno,
  ma lasciare accesso a chi estende la classe.

private

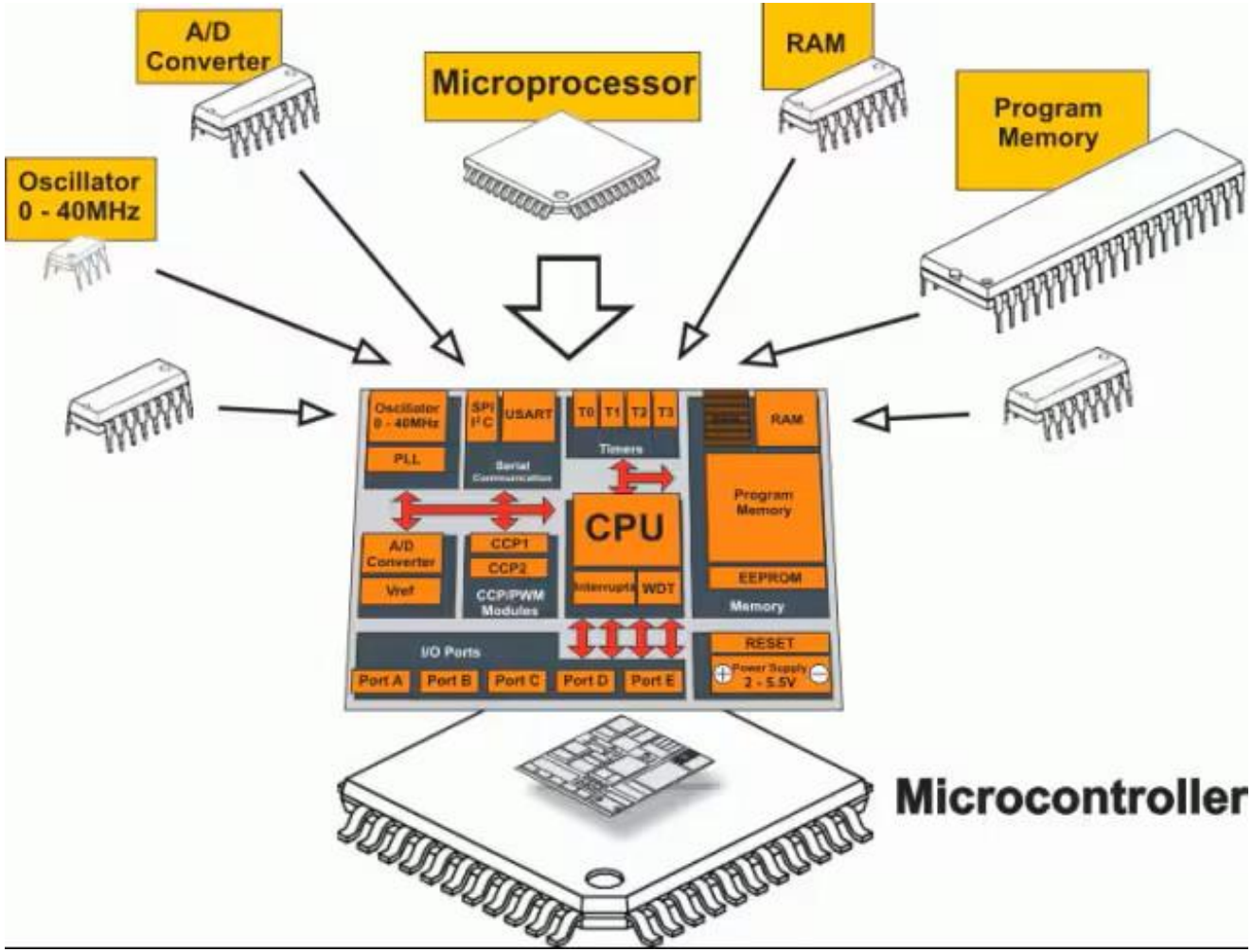
- Gli elementi private sono completamente nascosti:
  solo i metodi della stessa classe possono accedervi.
- Si usa per proteggere i dati interni e garantire che l'oggetto venga usato solo nel modo previsto.

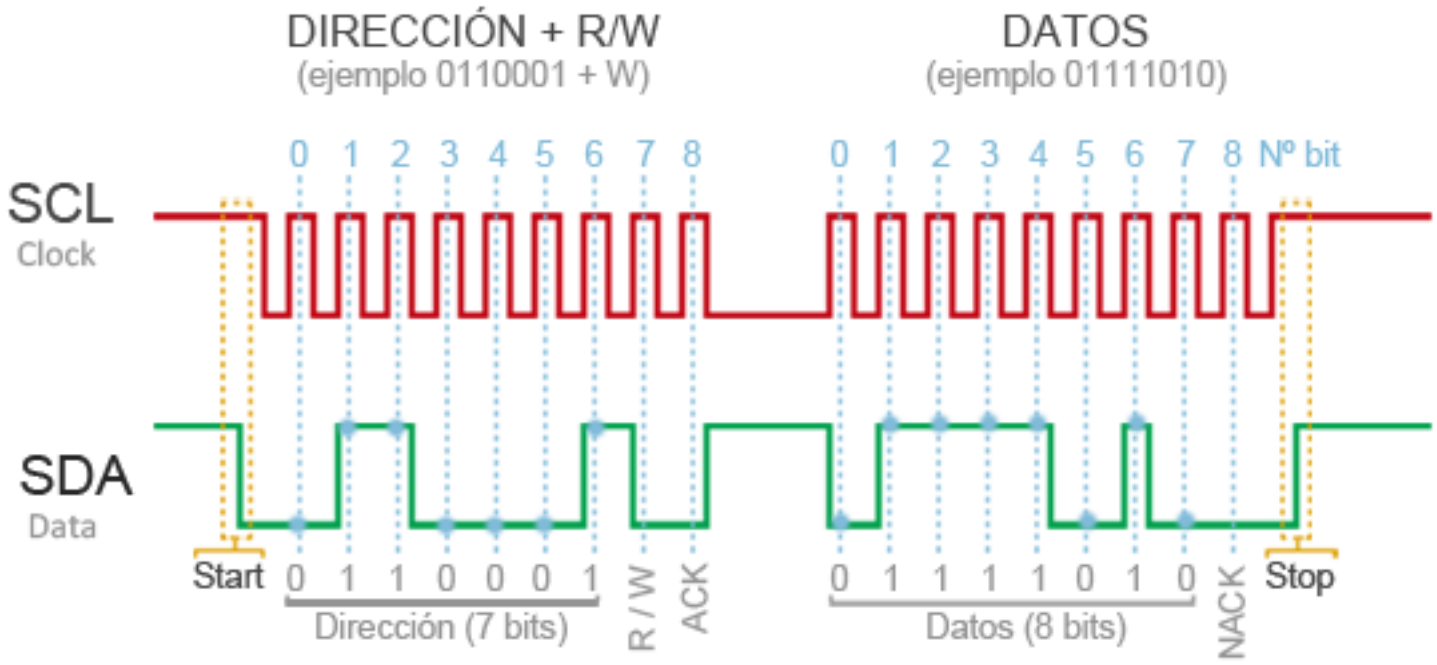
# WiFi & Bluetooth

https://docs.arduino.cc/language-reference/en/functions/wifi/overview/

https://docs.arduino.cc/learn/communication/bluetooth/

Domande, Dubbi, Perplessità?


A/D Converter
Microprocessor
RAM
Program Memory
Oscillator 0 - 40MHz
Oscillator 0 - 40MHz
PLL
USART
Timers
CPU
RAM
Program Memory
A/D Converter
CCP1
CCP2
Vref
WDT
EEPROM
Memory
I/O Ports
RESET
Port A
Port B
Port C
Port D
Port E
Power Supply 2 - 5.5V
Microcontroller


DIRECCIÓN + R/W
(ejemplo 0110001 + W)
DATOS
(ejemplo 01111010)
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 Nº bit
SCL
Clock
SDA
Data
Start
0 1 1 0 0 0 1
R / W
ACK
Dirección (7 bits)
0 1 1 1 1 0 1 0
NACK
Stop
Datos (8 bits)