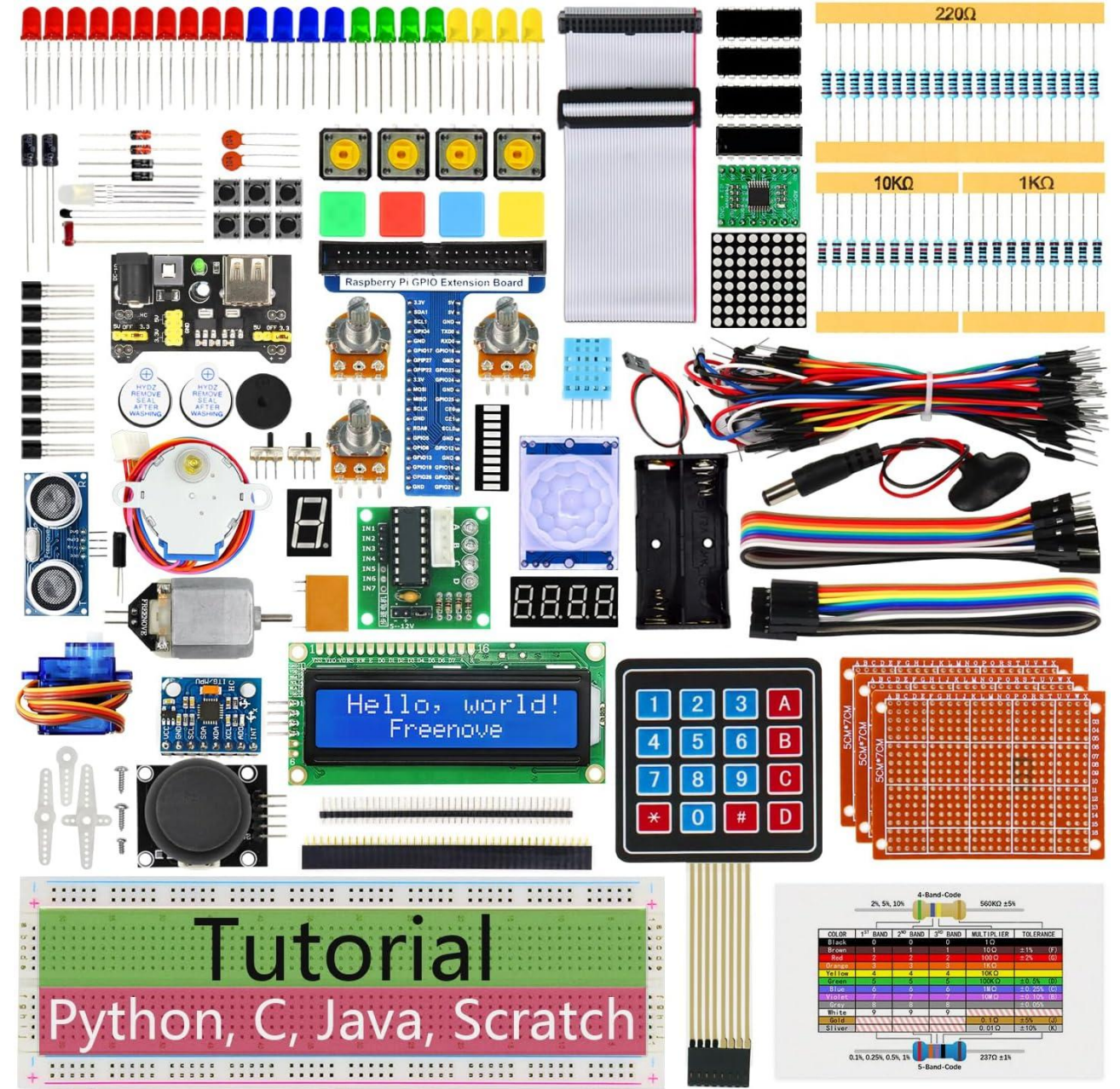


# CORSO AVANZATO DI INFORMATICA E ROBOTICA

LEZIONE 7: ARDUINO – OOP, attuatori, sensori di prossimità e LCD

# Cosa contiene il vostro kit?

<b>Raspberry Pi is NOT included in this kit!</b>  <b>Download Needed:</b> <b>Tutorial and Code</b> (No paper tutorial.)  The download link can be found on the box!							 x10 Red LED	 x4 Green LED	 x4 Blue LED	 x4 Yellow LED	 x1 RGB LED	 x1 LED Bar Graph
 x1 7-Segment Display	 x1 4-Digit 7-Segment Display	 x1 LED Matrix	 x20 Resistor-220	 x10 Resistor-1K	 x10 Resistor-10K		 x3 Potentiometer	 x2 Capacitor 0.1uF	 x2 Capacitor 10uF	 x6 Push Button	 x4 Big Push Button	 x1 Red Push Button Cap
 x2 Switch	 x1 Vibration Switch	 x1 Keypad	 x2 Rectifier Diode	 x2 Switch Diode	 x2 NPN Transistor	 x6 PNP Transistor	 x1 Yellow Push Button Cap					
 x1 2X AA Battery Holder	 x1 9V Battery Cable	 x1 Breadboard Power Module	 x1 Motor Driver Chip	 x3 Serial to Parallel Chip	 x1 ADC Module	 x2 Active Buzzer	 x1 Passive Buzzer					
 x1 Temperature and humidity Sensor	 x1 Thermistor	 x1 Photoresistor	 x1 Relay	 x1 Motor	 x1 Servo	 x1 Stepping Motor	 x1 Stepping Motor Driver					
 x1 Joystick	 x1 Infrared Motion Sensor	 x1 Ultrasonic Ranging Module	 x1 Accelerometer Module	 x1 LCD Module	 x1 GPIO Extension Board	 x1 Project Board	 x1 65 Jump Wire M-M					
 x1 10 Jump Wire F-F	 x1 10 Jump Wire F-M	 x1 40 Pin GPIO Cable	 x3 General Board	 x1 40 Pin Header	 x1 Female 40 Pin Header	 x1 Resistor Color Code Card	 x1 Plastic Box					

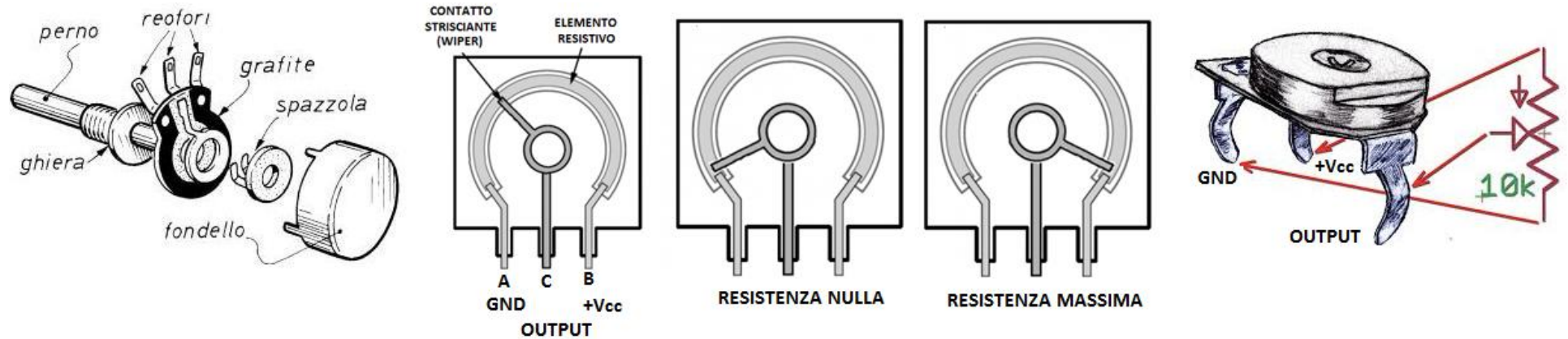


## GitHub con il codice del Corso

<https://github.com/CarloZambaldo/CorsoRobotica-Gonzaga>



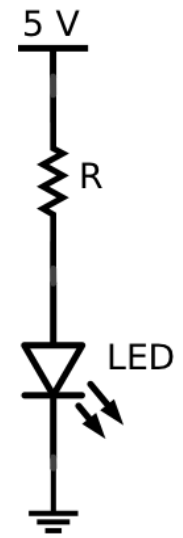
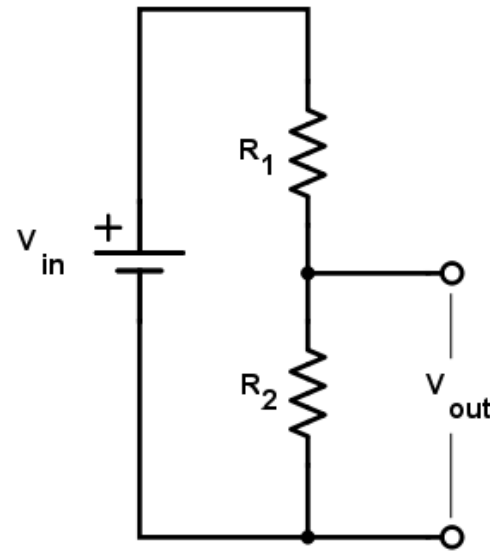
# Resistori in serie e in parallelo (richiami)



$$V = I \cdot R$$

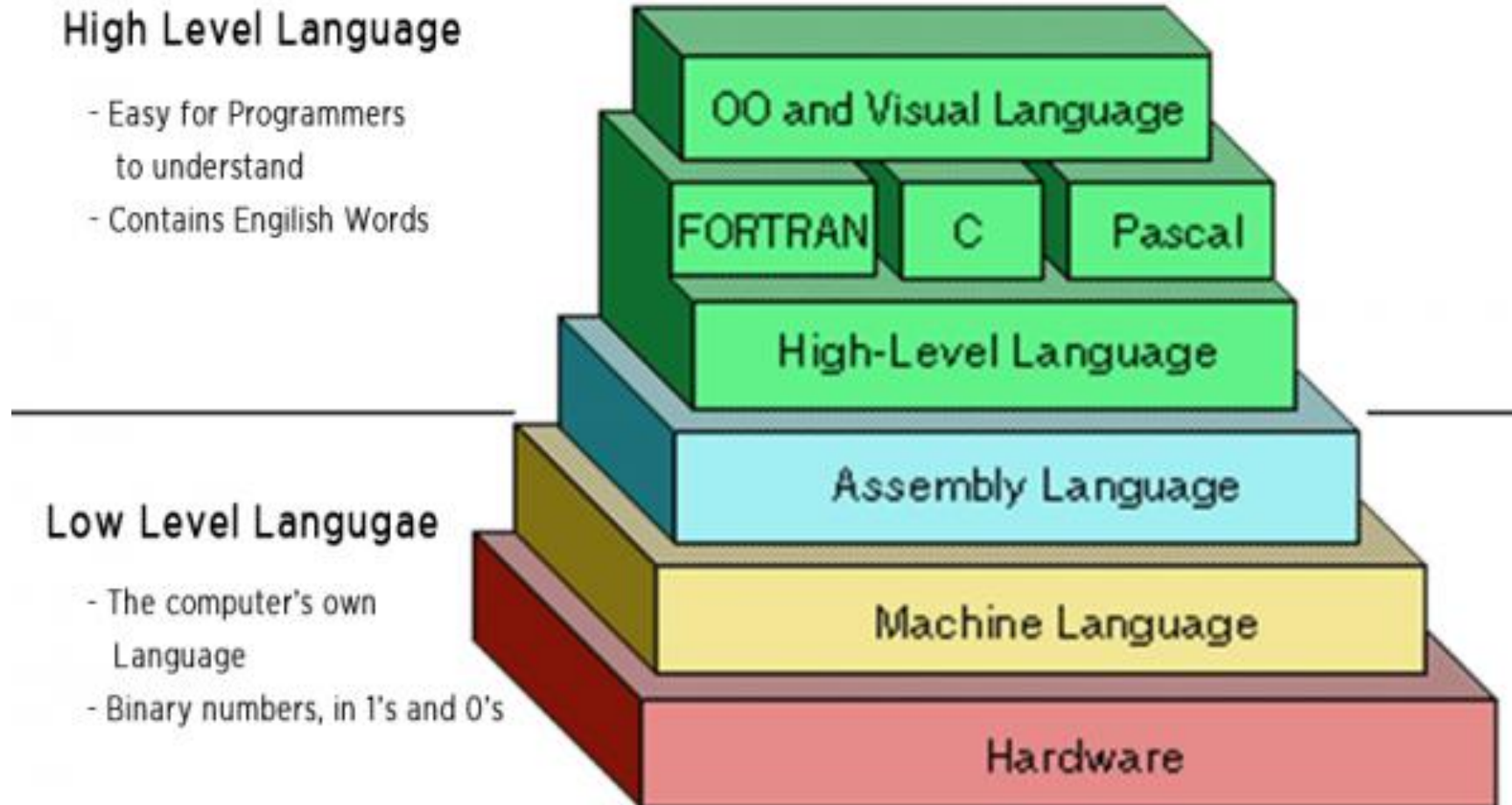
$$V_{out} = I \cdot R_2$$

$$I = \frac{V_{in}}{R_1 + R_2}$$



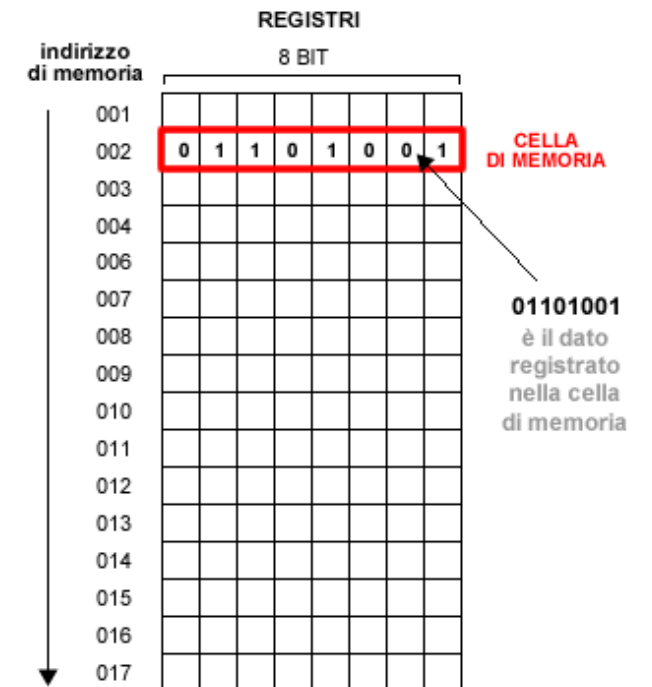


# Programmazione



# Tipi di variabili in C++/Arduino

- Interi (unsigned int, int, ...)
- Decimali (floating, double, ...)
- Caratteri (char)
- Vettori
  - Stringhe (vettori di caratteri: String)
  - Vettori di numeri
- Matrici («vettori di vettori»)
- Strutture dati (contengono vari tipi di dati)
- Liste
- Oggetti

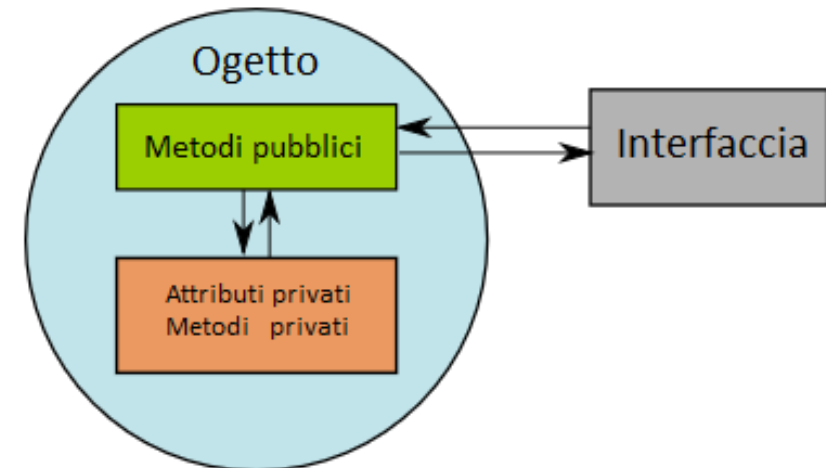
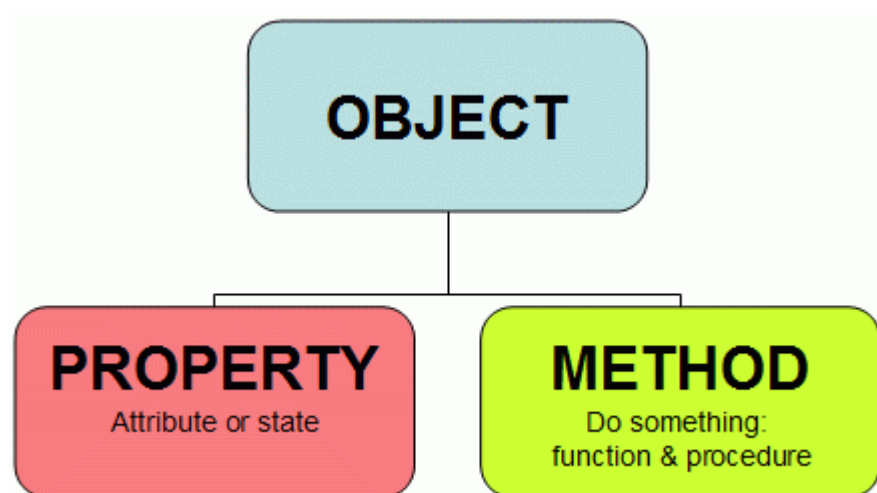


# Programmazione Orientata agli Oggetti (OOP)

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

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)



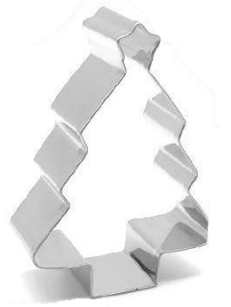
# Programmazione Orientata agli Oggetti (OOP)

«definisco» la struttura e 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 ALBERO { ... }
```

«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.



```
ALBERO frollina;
```



# 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 ( <code>moveMotor()</code> , <code>readSensor()</code> , 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() {        // metodo «accendi»  
            digitalWrite(pin, HIGH);  
        }  
        void spegni() {         // metodo «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: un esempio della libreria Arduino\_LED\_Matrix.h

```
#include "Arduino_LED_Matrix.h"
```

<https://docs.arduino.cc/tutorials/uno-r4-wifi/led-matrix/>

```
ArduinoLEDMatrix matrix; // crea un'istanza chiamata matrix della classe «ArduinoLEDMatrix»
```

```
const uint32_t happy[] = {
```

```
    0x19819,
```

```
    0x80000001,
```

```
    0x81f8000
```

```
};
```

```
void setup() {
```

```
    Serial.begin(115200);
```

```
    matrix.begin();           // uso il metodo «begin» della classe ArduinoLEDMatrix
```

```
    matrix.loadFrame(happy);  // uso il metodo «loadFrame» della classe
```

```
}
```

```
void loop() {
```

```
}
```



## OOP: public, protected and private modifiers

Modificatore	Visibile nella classe	Visibile nelle classi derivate	Visibile dall'esterno
public	✓	✓	✓
protected	✓	✓	✗
private	✓	✗	✗

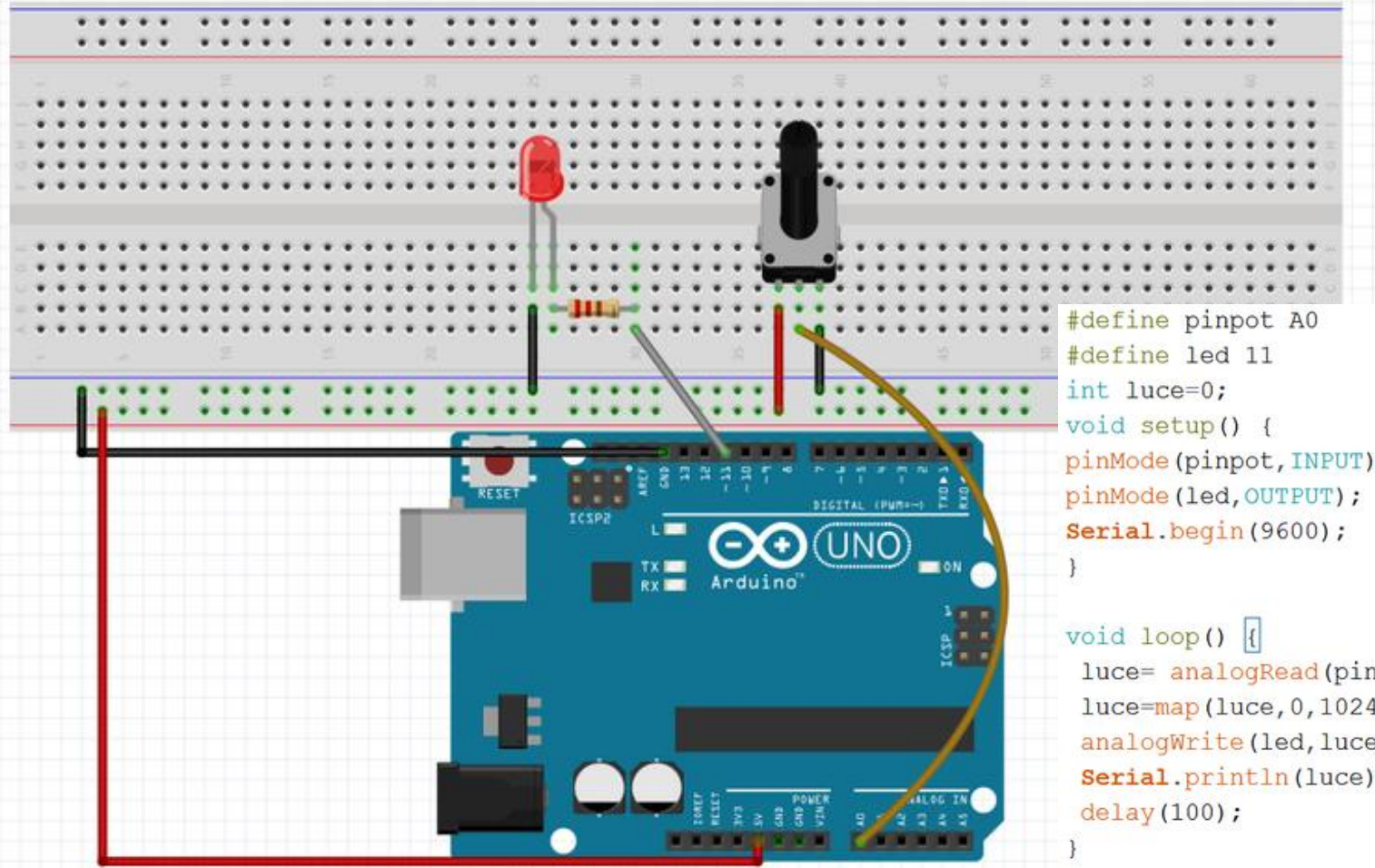
### protected

- 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.

## Esercizio 6 – LED «comandato» da un potenziometro



```
#define pinpot A0 //Definiamo il pin del potenziometro da 4,7k
#define led 11
int luce=0;
void setup() {
  pinMode(pinpot, INPUT);
  pinMode(led, OUTPUT);
  Serial.begin(9600); //Apriamo la comunicazione Seriale a 9600 baud
}

void loop() {
  luce= analogRead(pinpot); //Leggiamo il valore del potenziometro
  luce=map(luce, 0, 1024, 0, 255);
  analogWrite(led, luce);
  Serial.println(luce); // Scriviamo nel monitor seriale i dati ottenuti
  delay(100);
}
```

# Usare la matrice LED integrata

```
#include "Arduino_LED_Matrix.h"
```

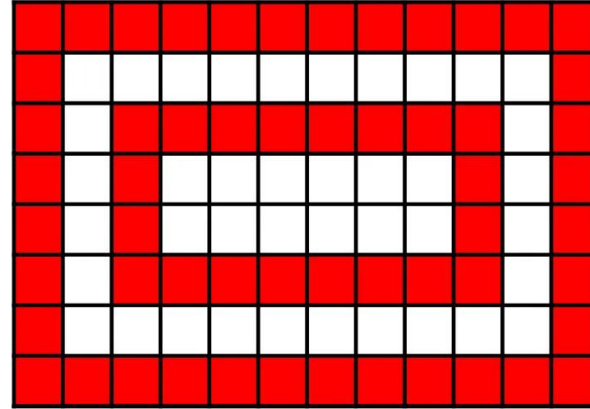
Da sapere:

```
matrix.loadFrame(nomeFrame);
```

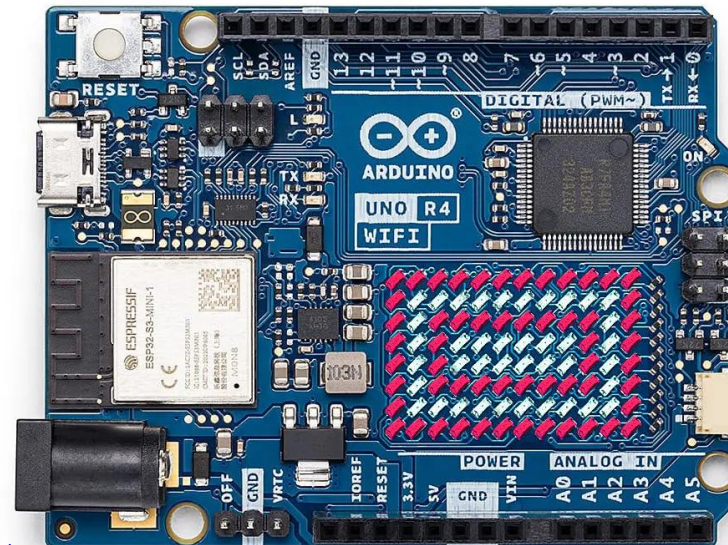
Esempio:

```
unsigned long frames[] = {
    0x3184a444,
    0x42081100,
    0xa0040000
};

for (int b = 0; b < 3; b++) {
    matrix.loadFrame(frames[b]);
}
```



```
byte frame[8][12] = {
    { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },
    { 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1 },
    { 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0 },
    { 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0 },
    { 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1 },
    { 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 },
    { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 }
};
```



<https://ledmatrix-editor.arduino.cc/>

<https://www.embedded.com/how-to-manage-the-led-matrix-in-arduino-r4/>

# Sensore Ultrasonico

Note:

`pulseIn(pin, valore)` Il valore per il sensore HC-SR04 deve essere HIGH

`pulseIn(pin, valore, timeout)` Timeout può essere omissso (di default è 1 secondo, 1.000.000  $\mu$ s).

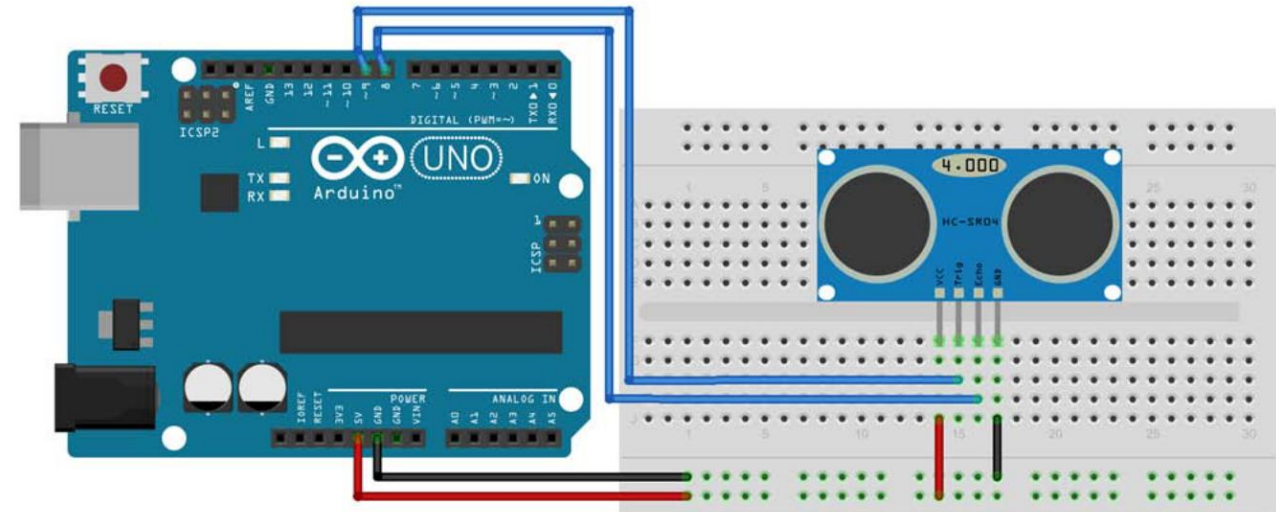
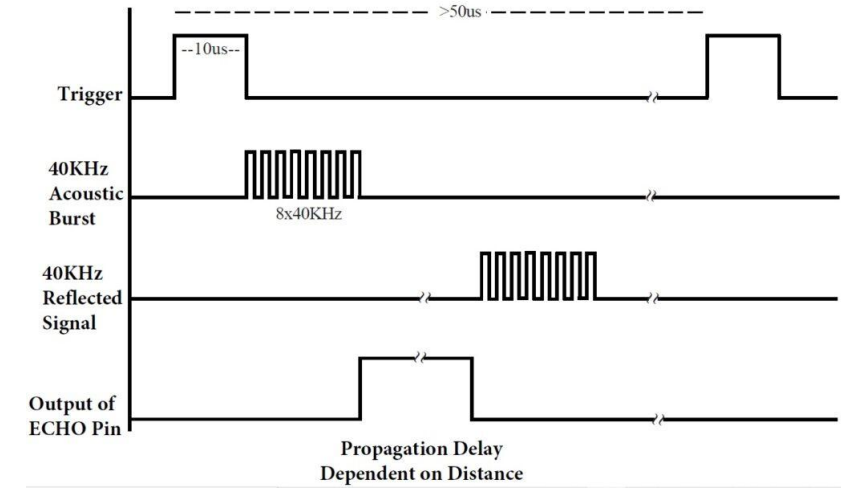
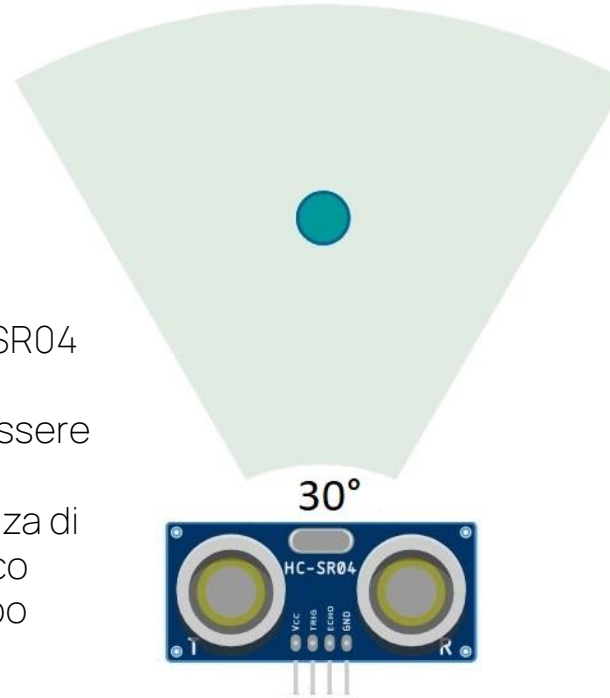
Si deve comunque considerare che la massima distanza di lavoro del sensore è 400 cm, che corrispondono a poco meno di 24 ms e che automaticamente il pin Echo dopo 38ms passa basso ed il segnale emesso deve essere considerato perso.

Inoltre come scritto nel Datasheet si devono aspettare almeno 50ms tra un invio di impulso ed un altro, per evitare che echi di vecchi impulsi siano erroneamente letti come validi e dare false letture.

Inoltre gli ostacoli da rilevare, oltre ad essere entro una distanza di 400 cm, devono essere all'interno di un cono di 30° come da foto

fonte: <https://www.makerslab.it/sensore-di-distanza-ad-ultrasuoni-hc-sr04-con-arduino/>

<https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf>





# Servo Motor

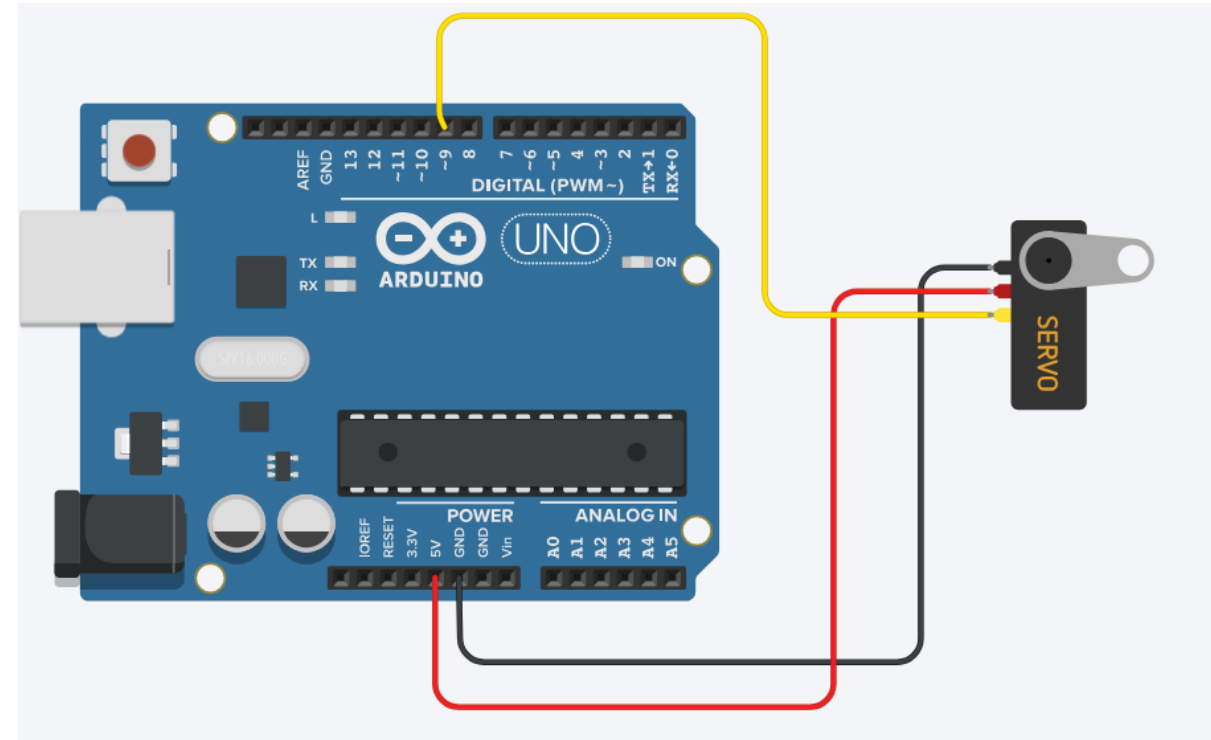
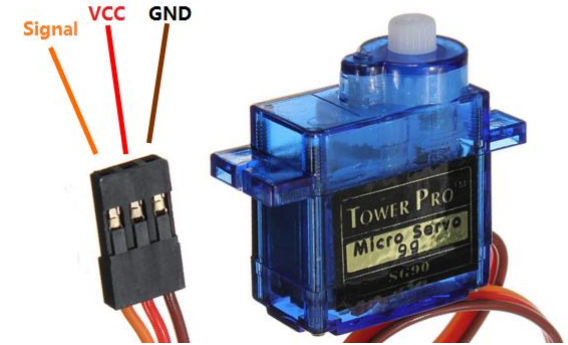
<https://www.tinkercad.com/things/7hd3jEjphsr>

```
#include <Servo.h> //Inserire la libreria Servo

Servo Servo1; //Il nome del servo è Servo1

void setup() {
  Servo1.attach (9); //Il Servo1 è collegato al pin
  digitale 9
}

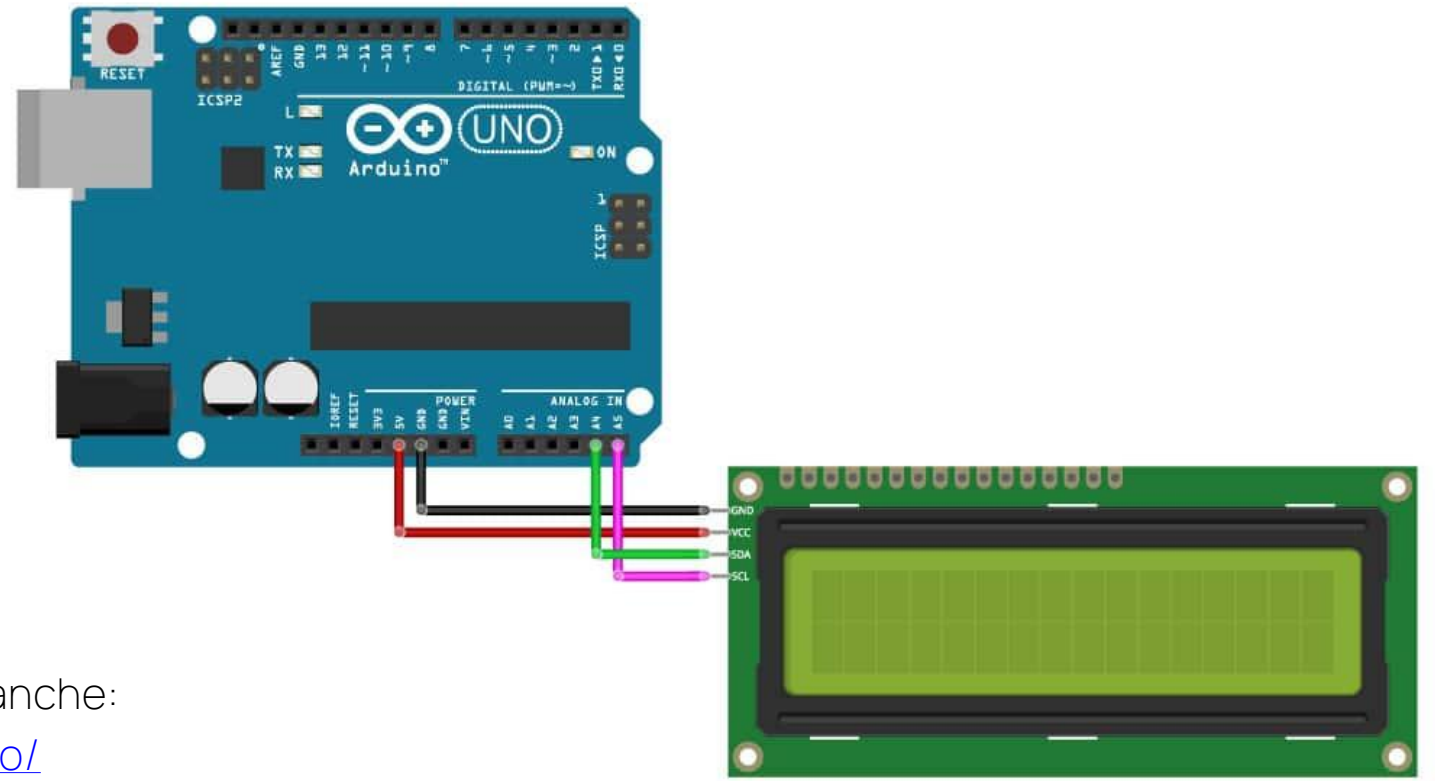
void loop() {
  //Imposto la posizione a cui si deve spostare
  Servo1.write (0);
  delay (500);
  Servo1.write (180);
  delay (500);
}
```





# LCD display Arduino

[https://www.handsontec.com/dataspecs/module/I2C\\_1602\\_LCD.pdf](https://www.handsontec.com/dataspecs/module/I2C_1602_LCD.pdf)



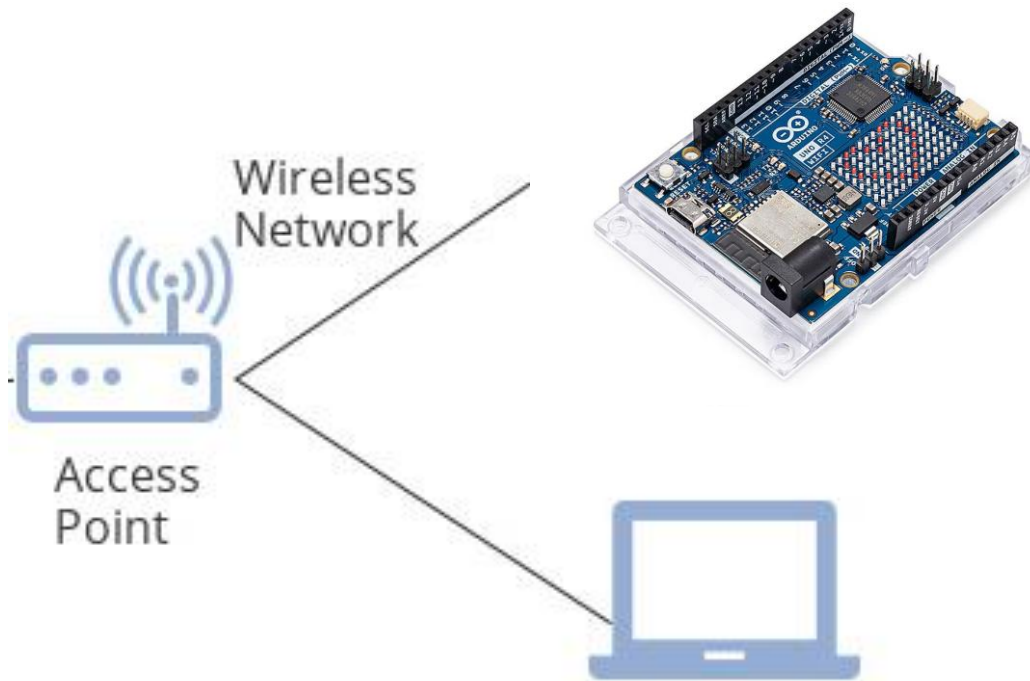
Per i metodi della libreria LiquidCrystal\_I2C vedi anche:

<https://www.make-it.ca/i2c-lcd-display-on-arduino/>

# WiFi & Bluetooth

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

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





Domande, Dubbi, Perplessità?





### Traditional floating point (IEEE 754 style)

