

CORSO AVANZATO DI INFORMATICA E ROBOTICA

LEZIONE 4: ARDUINO e C

Codice C

Questo codice stampa i primi N numeri pari, dove N è inserito dall'utente

```
#include <stdio.h>

int main() {
    int N;

    printf("Inserisci un numero: ");
    scanf("%d", &N);

    printf("Numeri pari tra 1 e %d:\n", N);

    for (int i = 1; i <= N; i++) {
        if (i % 2 == 0) {
            printf("%d ", i);
        }
    }

    printf("\n");
    return 0;
}
```

GitHub con il codice del Corso

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



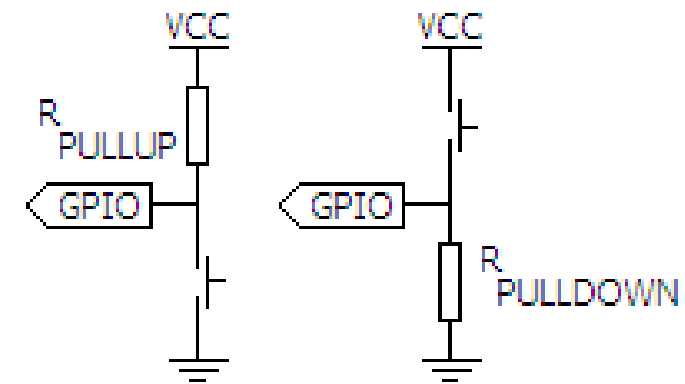
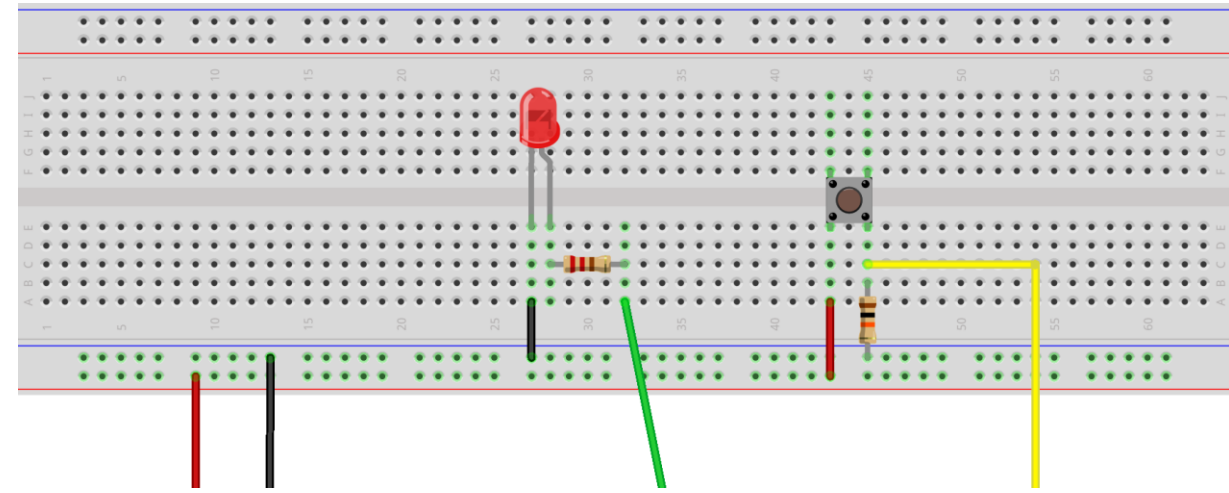
Esercizio 2 – Aggiungere un Pulsante

```
#define LED_PIN 13
#define BUTTON_PIN 2

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

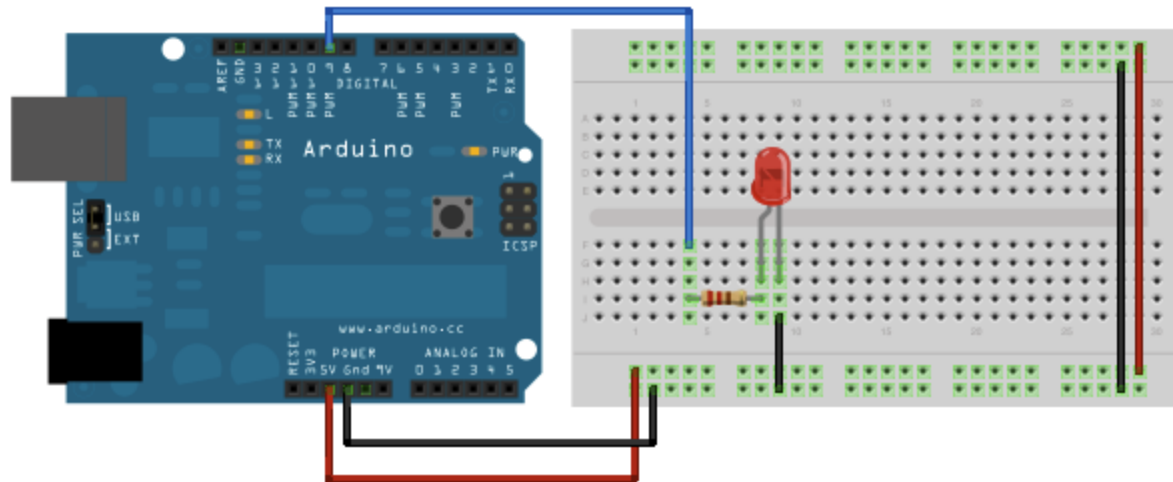
void loop() {
    int buttonState = digitalRead(BUTTON_PIN);
    if (buttonState == HIGH) {
        digitalWrite(LED_PIN, HIGH);
    } else {
        digitalWrite(LED_PIN, LOW);
    }
}
```

Nota: generalmente il resistore per i LED è 220Ω , mentre Pull-down si usa $1k\Omega$



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

Esercizio 4 – LED a luminosità variabile



```

/*
  Fade

  This example shows how to fade an LED on pin 9
  using the analogWrite() function.

  This example code is in the public domain.
  */

int led = 9;          // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}

```

Esercizio 5 – fotoresistenza

```
#define LDR_PIN A0
```

```
void setup() {  
  // put your setup code here, to run once:  
  Serial.begin(9600);  
}
```

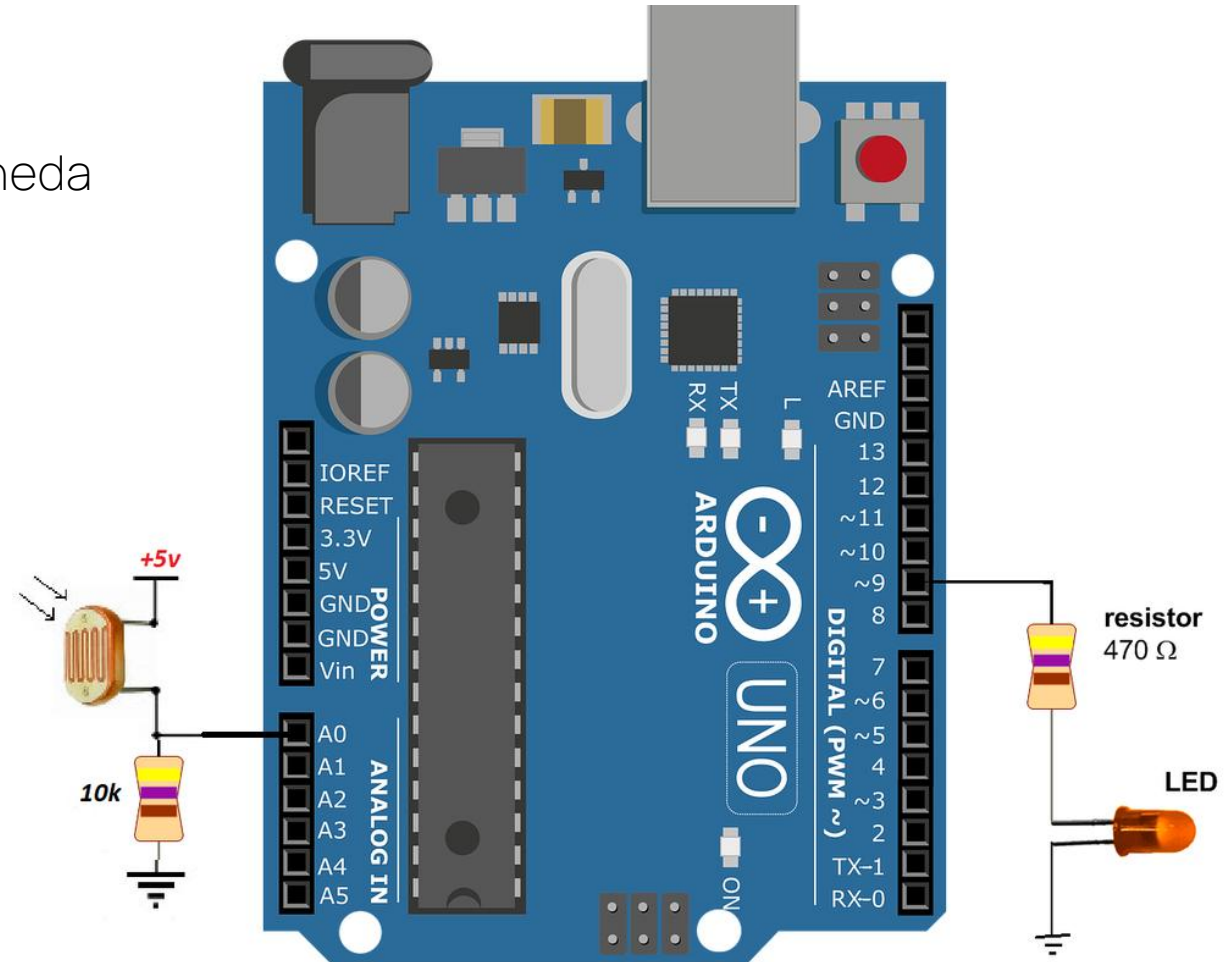
```
void loop() {  
  
  float luminosita = misuraLuminosita();
```

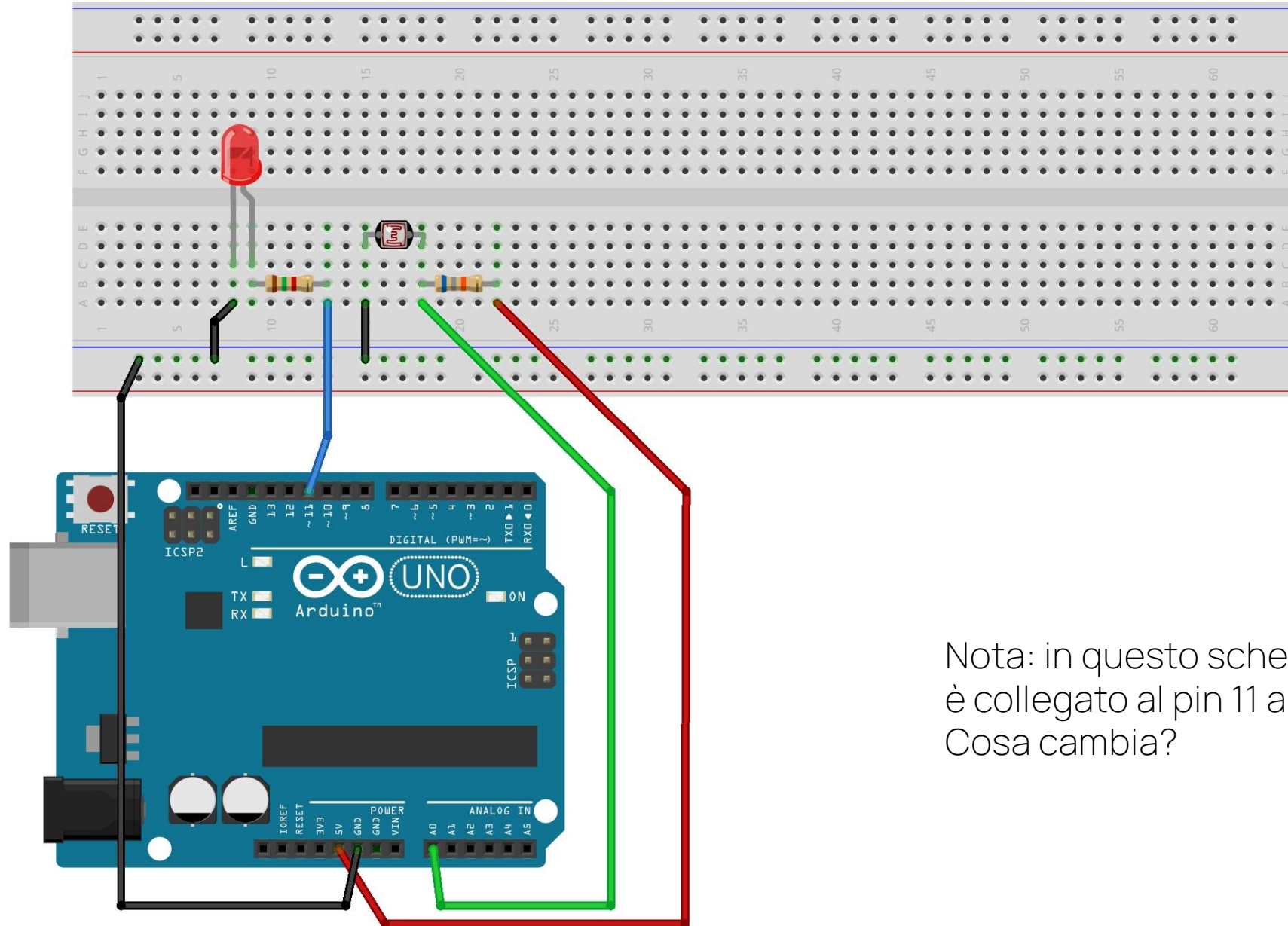
```
  Serial.print("Lumunosità: ");  
  Serial.println(luminosita);  
  delay(1000);  
}
```

```
float misuraLuminosita() {  
  int valore_ldr = analogRead(LDR_PIN);  
  float val_ldr_convertito = map(valore_ldr, 20, 1005, 0, 100);  
  
  return val_ldr_convertito;  
}
```

Per creare il circuito ci serviranno:

- una fotoresistenza
- una resistenza da 1k ohm
- una breadboard
- tre cavetti jumper
- una scheda Arduino
- un cavo usb C





Nota: in questo schema il LED
è collegato al pin 11 anziché 9...
Cosa cambia?

Usare la matrice LED integrata

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

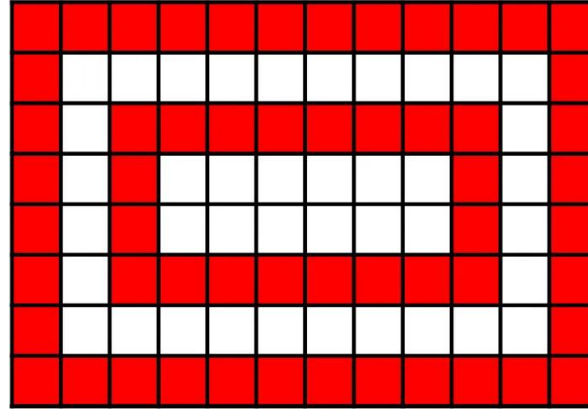
Da sapere:

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

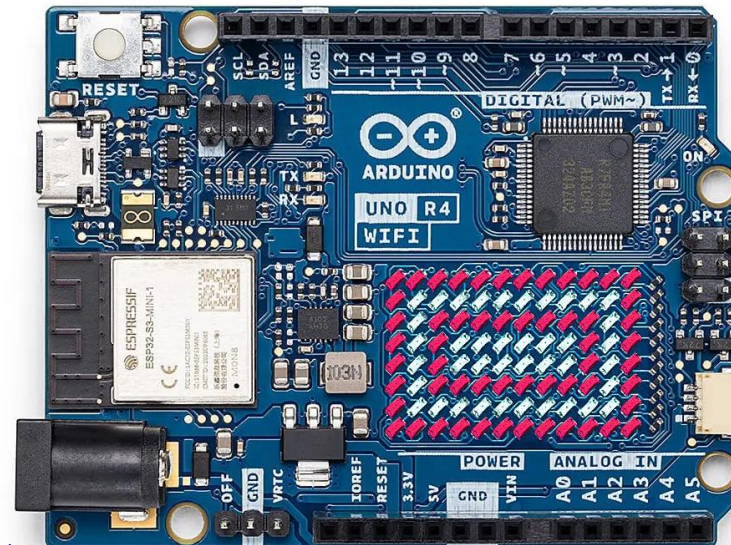
Esempio:

```
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 }
};
```



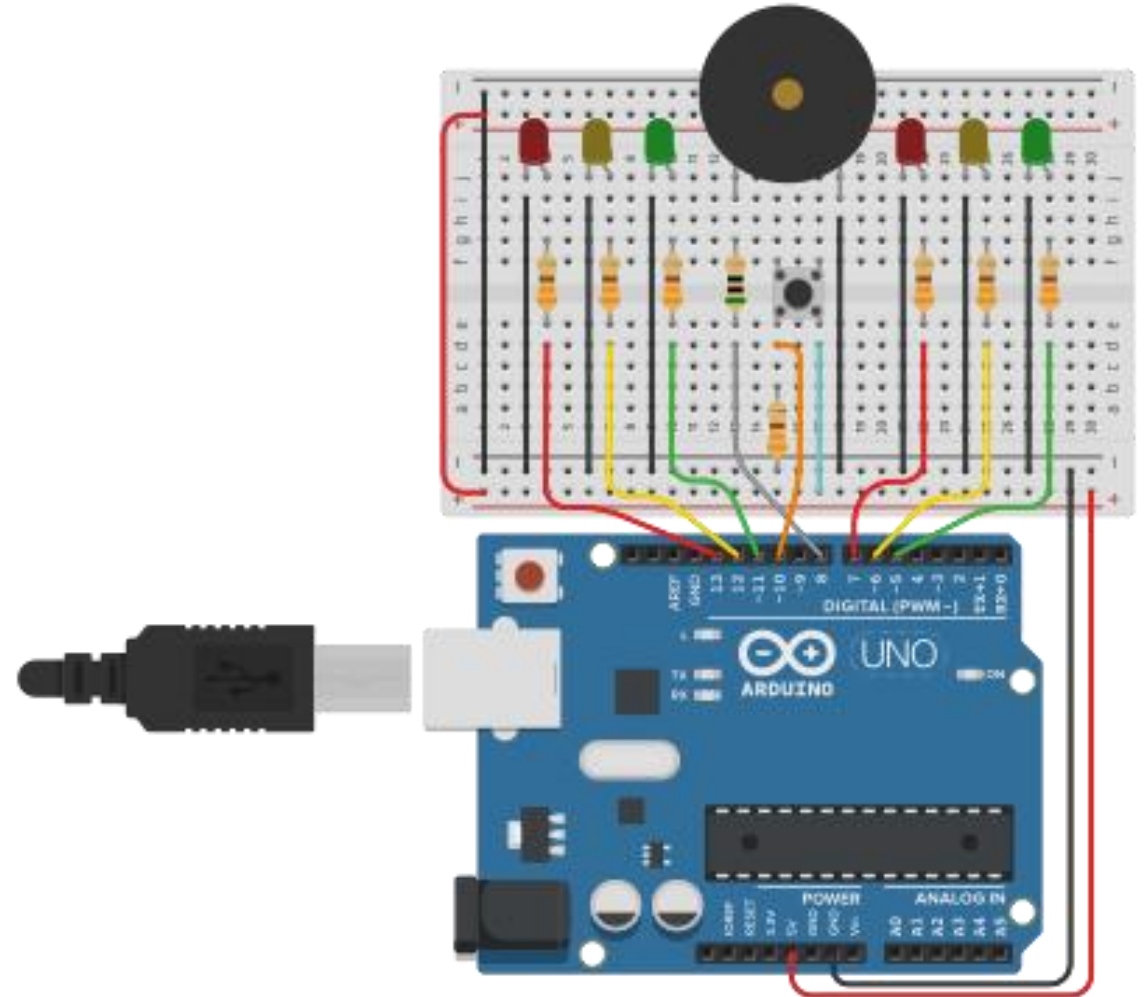
<https://ledmatrix-editor.arduino.cc/>

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

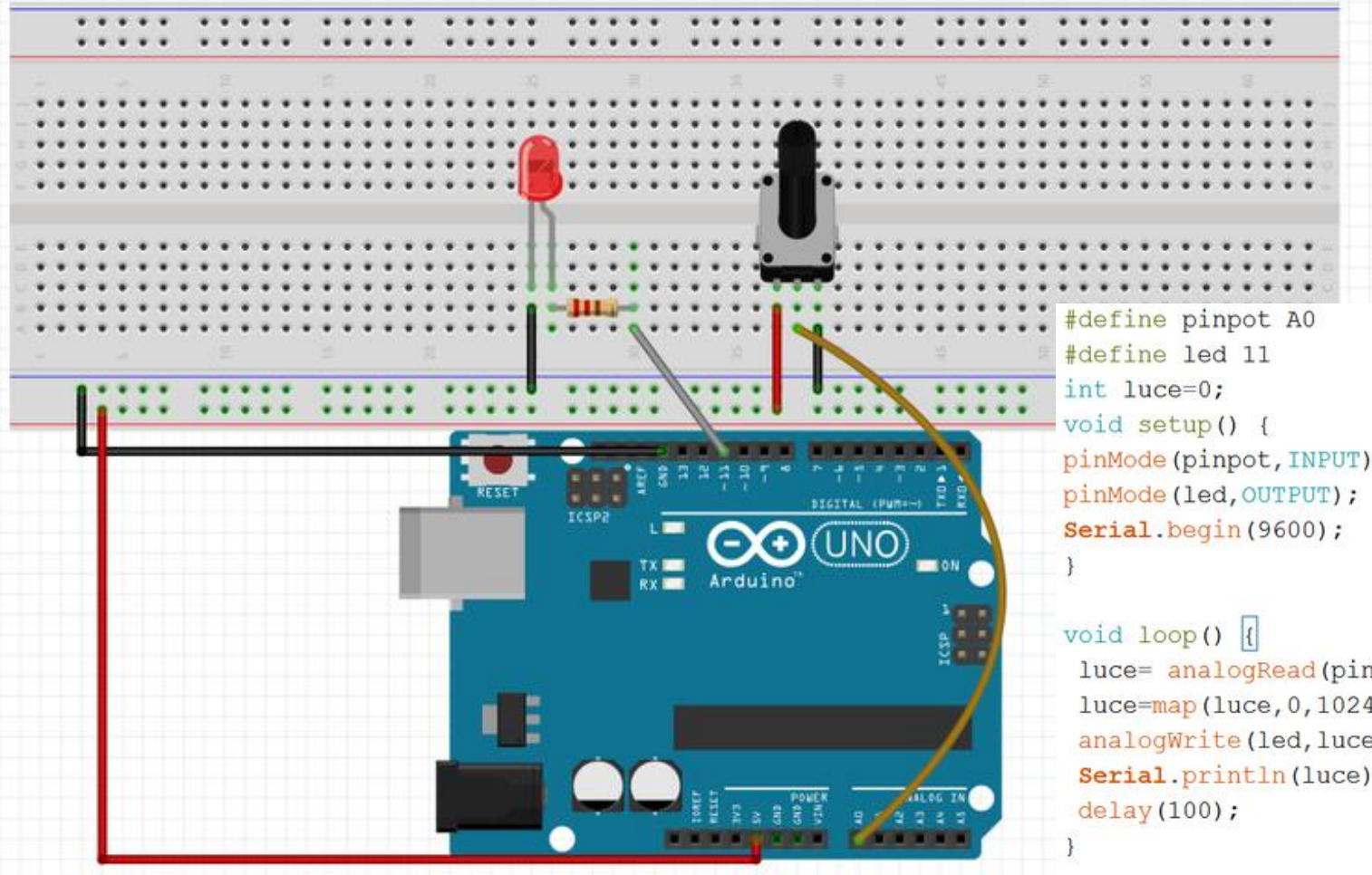
Esercizio 3 – Semaforo controllato da pulsante di richiesta e segnale sonoro

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.



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
}

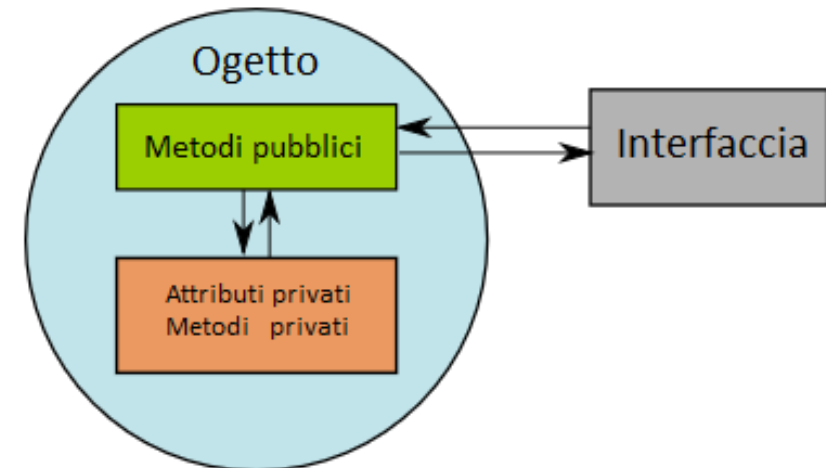
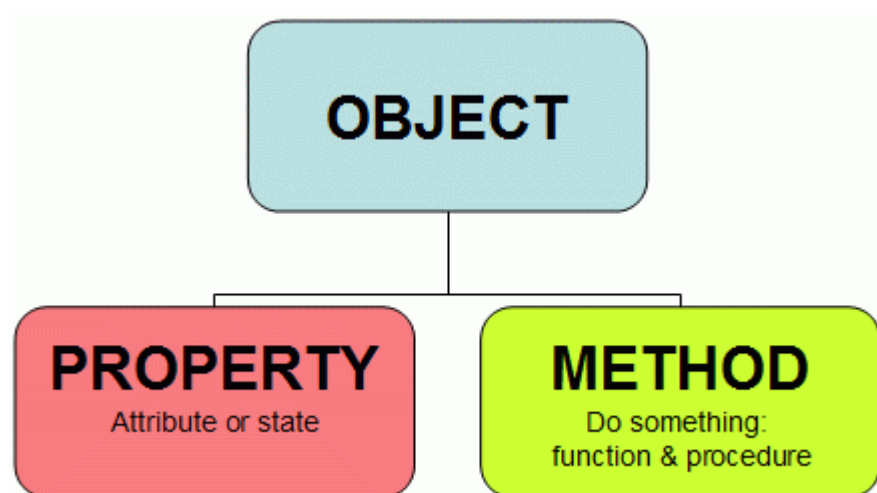
void loop() {
  luce= analogRead(pinpot); //Leggiamo il valore del potenziometro
  luce=map(luce, 0, 1024, 0, 255);
  analogWrite(led, luce);
  Serial.println(luce); // Scriviamo nel monitor seriale i dati ottenuti
  delay(100);
}
```

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 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: 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.

WiFi & Bluetooth

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

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



Domande, Dubbi, Perplessità?