

Arduino e Processing

Come interfacciarsi con il mondo reale



Parte I

Arduino

Cos'è Arduino

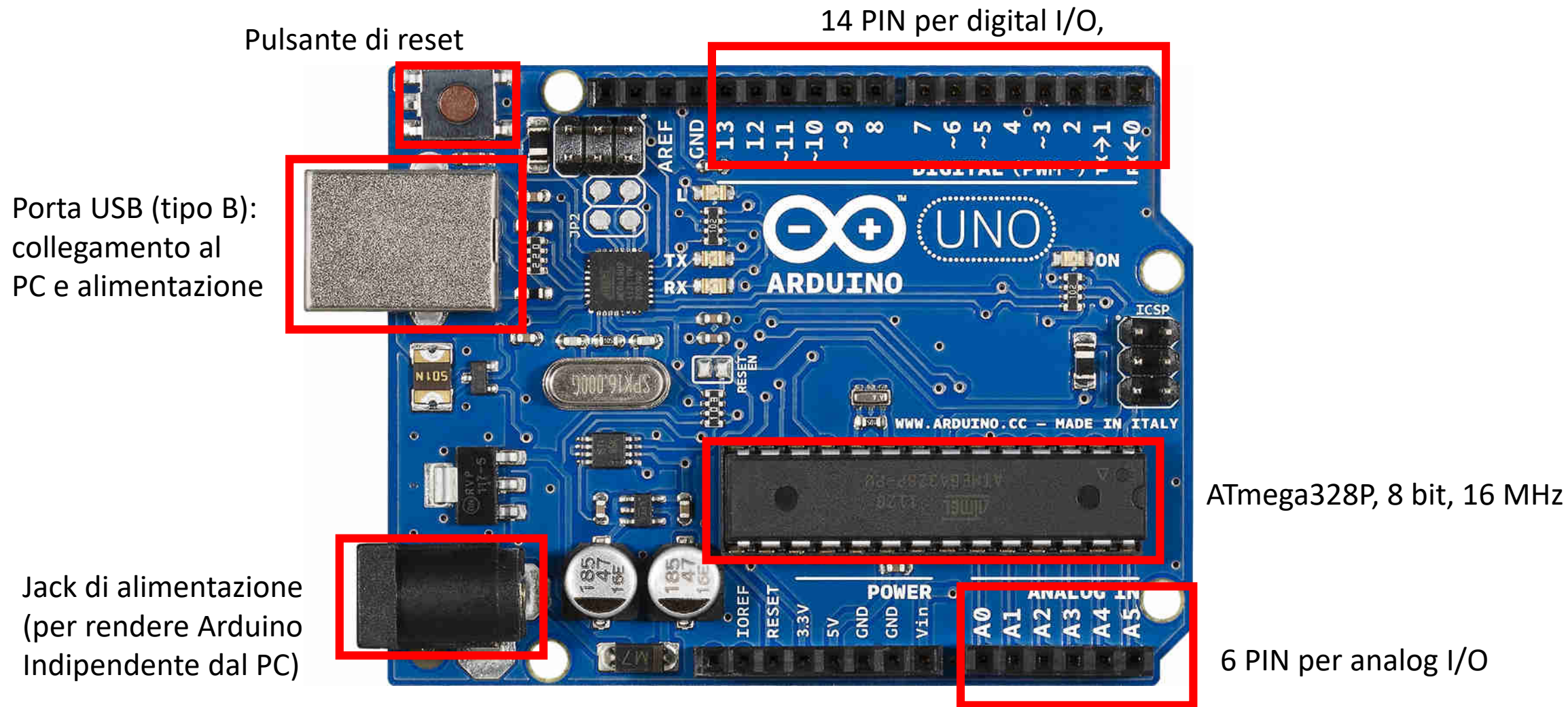
- Arduino (<https://www.arduino.cc/>) è una piattaforma **open-source** di **prototipazione elettronica** (nato ad opera di italiani ad Ivrea, nel **2003**).
- È stata ideata per rendere facile l'interazione di un sistema di calcolo con l'ambiente circostante utilizzando una grande varietà di sensori, motori ed altri attuatori.
- Il microprocessore sulla scheda si programma mediante un insieme di funzioni C/C++ usando il linguaggio ed ambiente di sviluppo integrato **Wiring** (a sua volta basato su **Processing**).
- I progetti sviluppati con Arduino
 - possono funzionare controllati direttamente dal software sulla scheda (modalità **stand-alone**),
 - oppure possono comunicare con software in esecuzione su un computer (per esempio **Processing**).
- Esistono molte versioni della scheda Arduino e molti prodotti basati su di essa:
 - <https://www.arduino.cc/en/Main/Products>

Programmare Arduino

- Arduino si può programmare tramite l'apposito IDE, scaricabile da:
 - <https://www.arduino.cc/en/Main/Software>
- Nei successivi esempi programmeremo la board prima con **l'IDE di Arduino** e poi direttamente da **Processing**, senza scrivere codice nativo.
- Anche per la seconda modalità l'IDE di Arduino va comunque installato perché porta con sé i driver della scheda ed è utile per installare il programma che implementa il protocollo di interfaccia via seriale verso Processing.

La scheda (Arduino UNO)

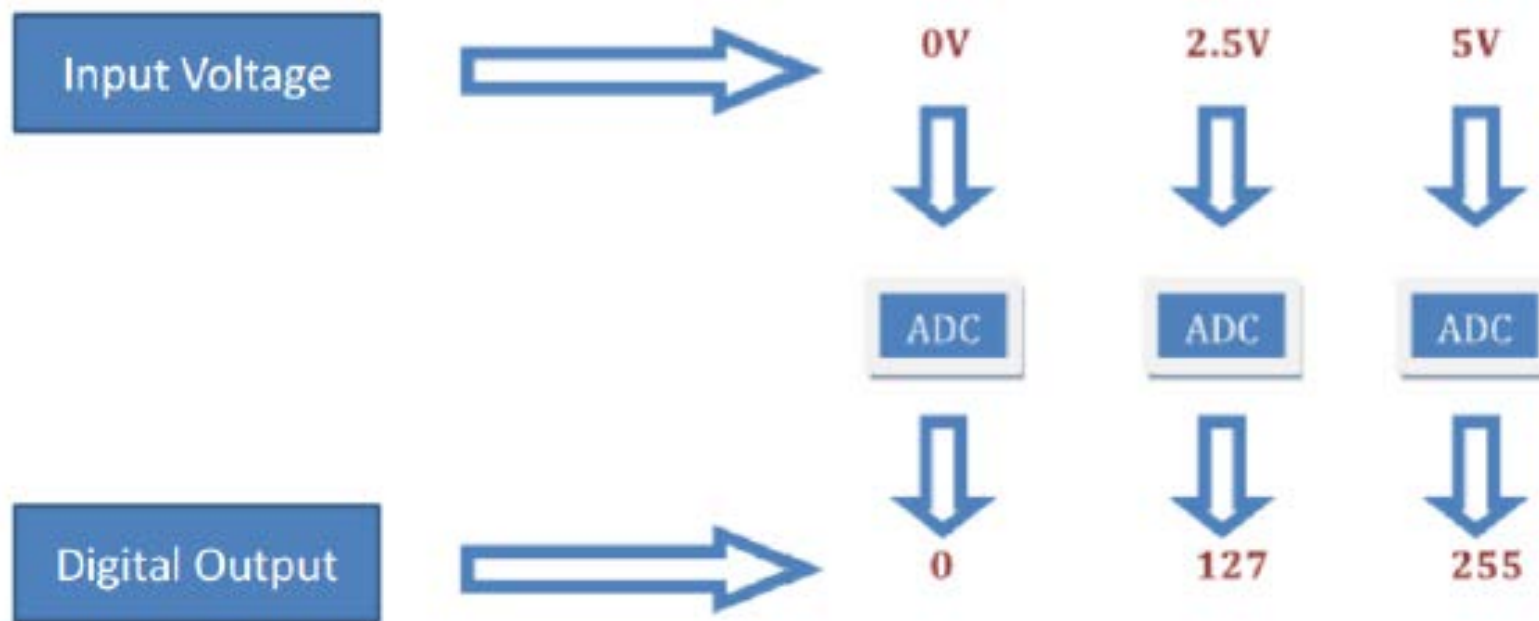
Prodotto a partire dal 2010
Microcontroller: ATmega328
14 digital input/output (I/O) pins
6 analog input pins



Segnali digitali e analogici

- Un **segnale analogico** è un **segnale continuo** e la sua caratteristica variabile è una rappresentazione di un'altra quantità variabile nel tempo.
- Un **segnale digitale** utilizza **valori discreti**. Sebbene le rappresentazioni digitali siano discrete, le informazioni rappresentate possono essere discrete o continue.
- Per l'utilizzo con Arduino bisogna **convertire i segnali continui in segnali digitali** (tramite l'uso di **ADC: Analog to Digital Converter**).

ADC



6 PIN per ADC

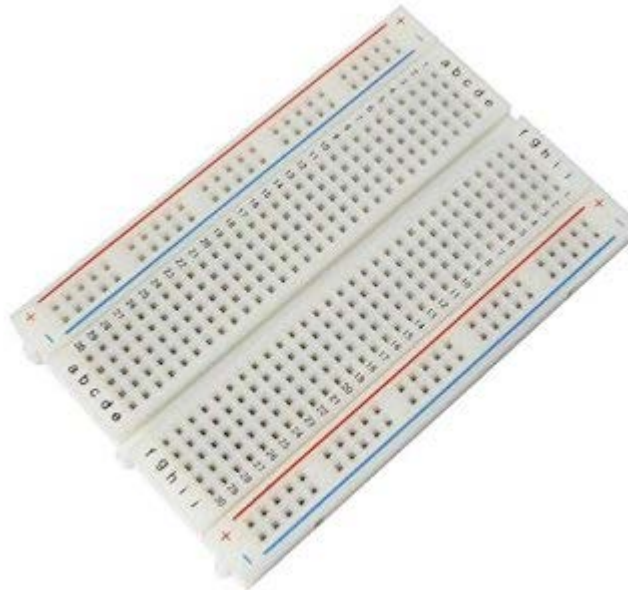
La scheda Arduino Uno contiene 6 pin per ADC.
La risoluzione della conversione da analogico a digitale è di 10 bit.
Ciò significa che le tensioni di ingresso tra 0V e 5V sono mappate in valori interi compresi tra 0 e 1023.

La Breadboard

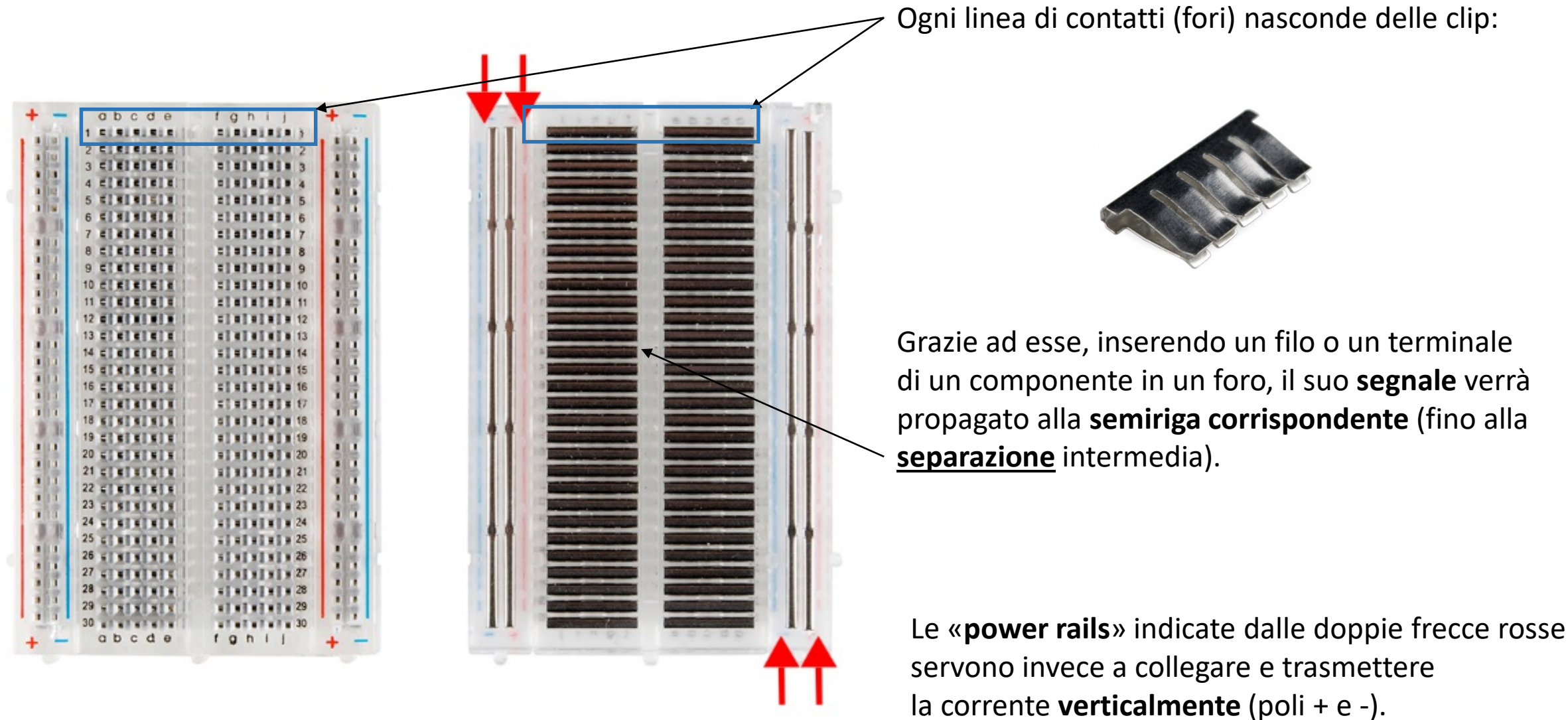
- Origine del nome:



- Una breadboard moderna:



La Breadboard



Esempi di elementi collegabili alla breadboard ed alle porte di Arduino

Cavo/jumper

Alimentatore/Pila

Resistore/Fororesistore/Termistore

Condensatore

Induttore

Diodo

LED

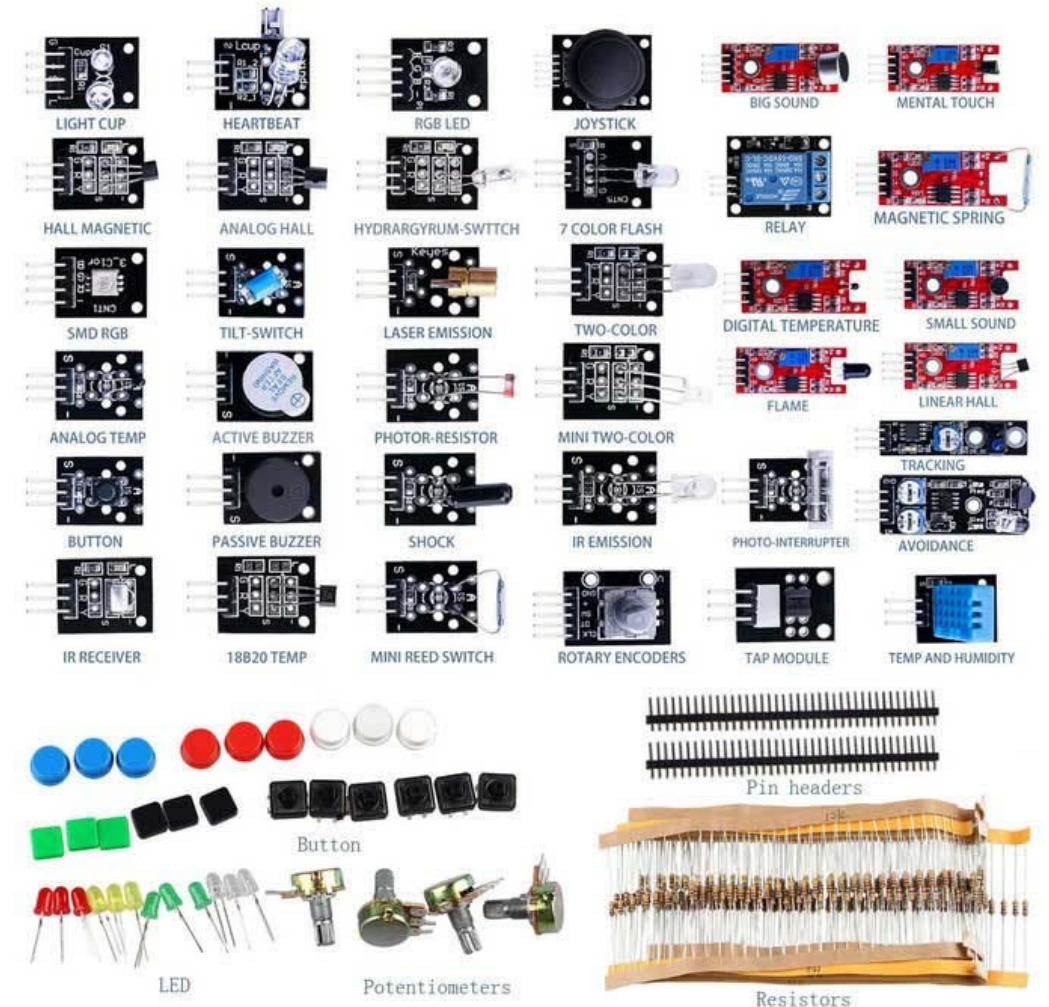
Pulsante

Transistor

Relé

Servomotore

...



L'ambiente di sviluppo (IDE)

Quando Arduino viene acceso/resettato esegue il metodo `setup()`, che serve per compiti di inizializzazione.

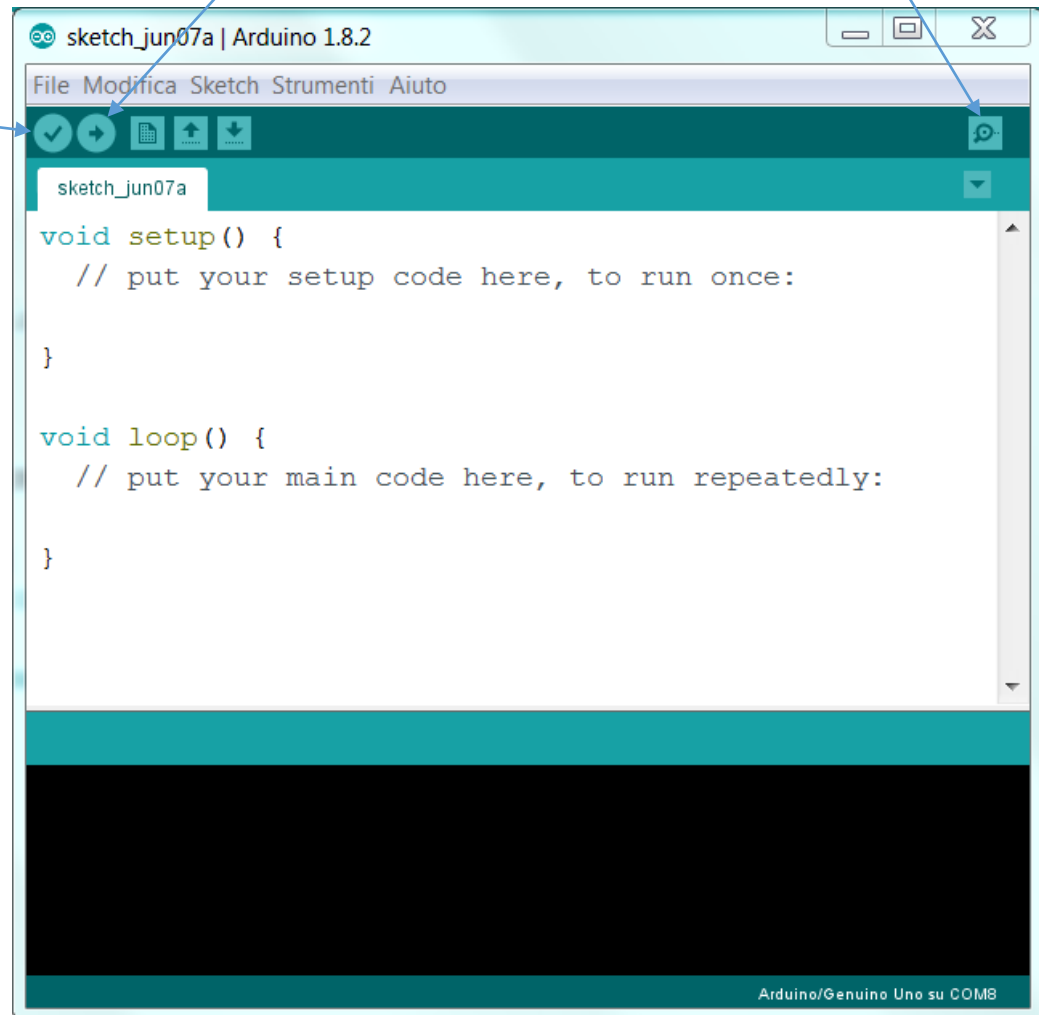
Dopo la fase di inizializzazione i comandi contenuti nel metodo `loop` vengono eseguiti in modo continuato (fino allo spegnimento/reset della scheda).

Pulsante di compilazione/verifica del codice.

Pulsante per il caricamento del codice sulla scheda.

Monitor della porta seriale (USB)

Console per messaggi, output, ...



Come rappresentare i PIN per l'I/O

- Nel codice, per agire su un particolare PIN di I/O, è sufficiente agire come segue:
 - PIN digitali: si utilizza un intero da 0 a 13, sia in ingresso che in uscita;
 - PIN analogici: si utilizza **A0, A1, A2, A3, A4, A5**, generalmente in ingresso.
- Valori:
 - PIN digitali: **LOW** (0 V) oppure **HIGH** (5 V);
 - PIN analogici: un intero da 0 a 1023.
- Ad esempio il codice seguente imposta il PIN digitale 12 per l'output ed invia ad esso il segnale HIGH (5V):

```
pinMode(12, OUTPUT);  
digitalWrite(12, HIGH);
```

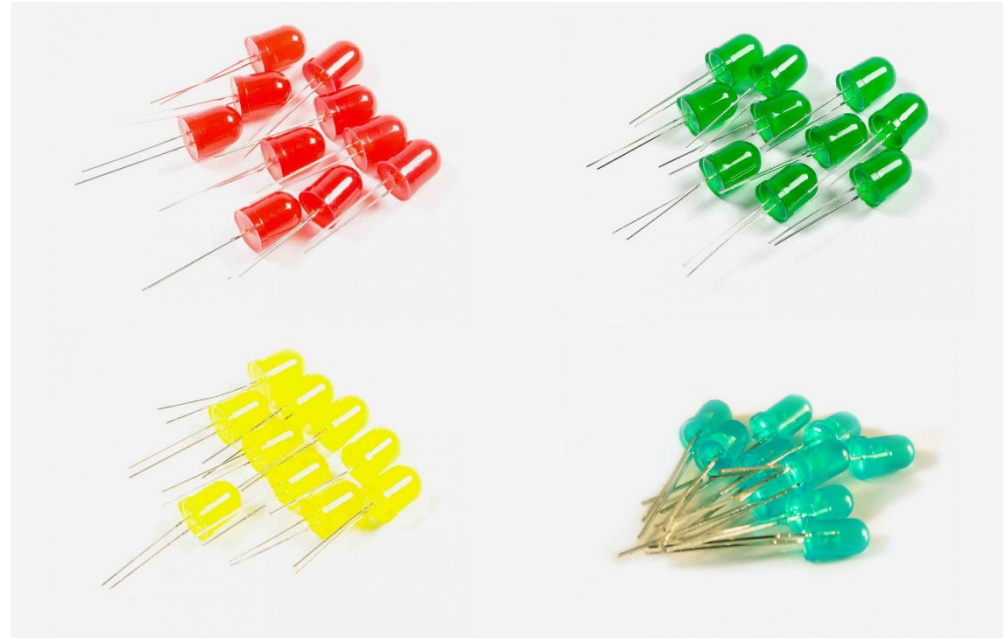
Libreria di Arduino

- La documentazione di riferimento del linguaggio è reperibile all'indirizzo <https://www.arduino.cc/reference/en/>
- Le due funzioni che devono essere implementate sono **setup()** e **loop()**.
- Le funzioni di libreria principali usate negli esempi sono:
 - **Serial.begin()**
 - **Serial.print()/Serial.println()**
 - **pinMode()**
 - **digitalRead()/digitalWrite()**
 - **analogRead()/analogWrite()**
 - **tone()/noTone()**
 - **Servo motor** → **motor.attach()** e **motor.write()**
 - **map()**
 - **delay()**

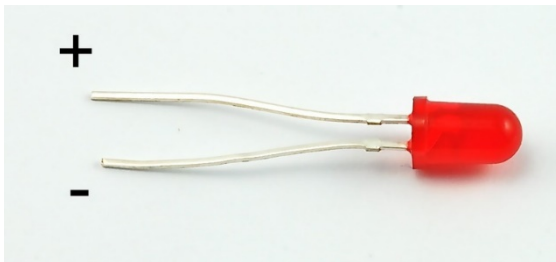
Setup e Loop

```
void setup()  
{  
  // initialization of variables, pin modes, libraries  
  // run once after each power up or reset  
}  
void loop()  
{  
  // loops the content consecutively  
  // allowing the program to change and respond  
}
```


II LED



- Il **LED** (Light Emitting Diode) è un dispositivo che si serve della capacità di alcuni materiali semiconduttori di **produrre fotoni** attraverso un fenomeno di emissione spontanea.
- L'estremità più corta del LED rappresenta il **catodo** (polo negativo), mentre la più **lunga** rappresenta l'**anodo** (polo positivo):



Il LED on-board di Arduino (sketch: LED_13.ino)

- La **porta digitale 13** (PIN 13) permette anche di interagire con il vicino **LED interno** alla scheda.
- Il codice seguente **alterna** per il LED interno dei cicli di accensione (per 5 secondi) e di spegnimento (per 5 secondi):

```
#define LED_PIN 13
#define TIME_ON 5000
#define TIME_OFF 5000

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

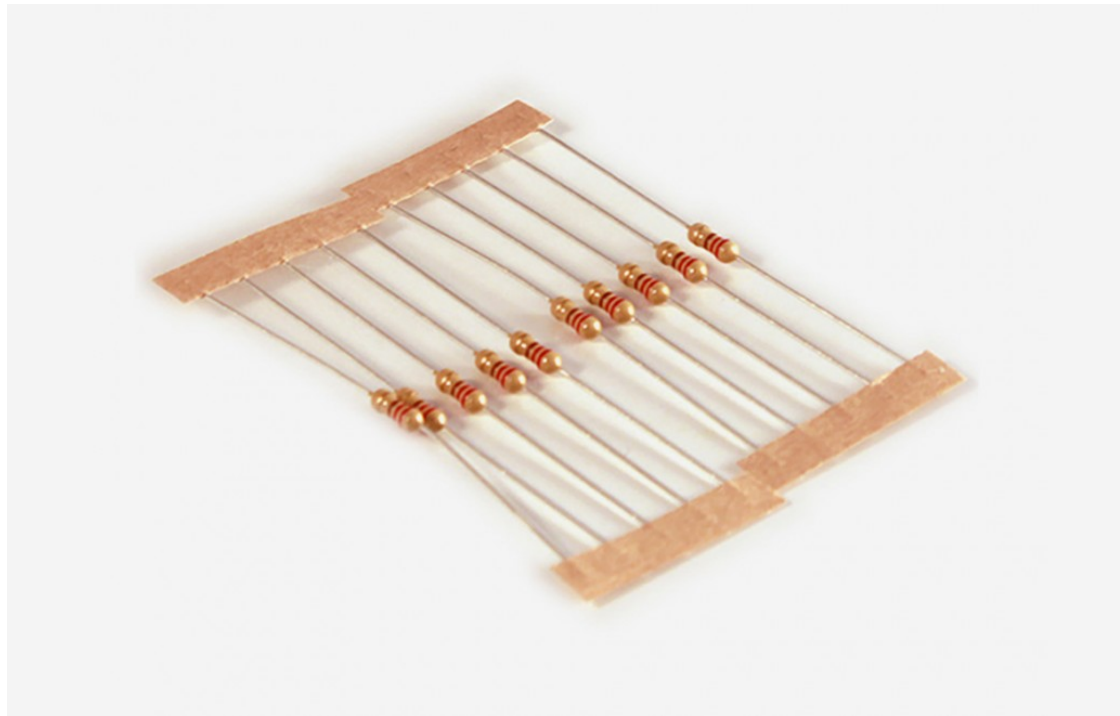
void loop() {
  digitalWrite(LED_PIN, HIGH);
  delay(TIME_ON);
  digitalWrite(LED_PIN, LOW);
  delay(TIME_OFF);
}
```



LED "on-board"

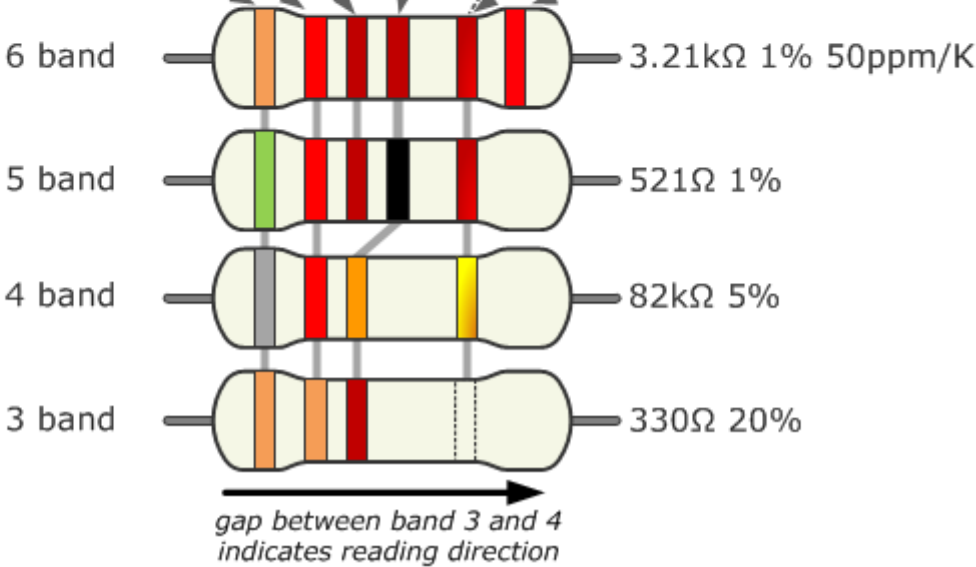
Resistori

- Il resistore è un componente elettrico che oppone una specifica resistenza elettrica (unità di misura: Ohm, Ω) al passaggio della corrente elettrica.



Resistenze – tabella dei codici colore

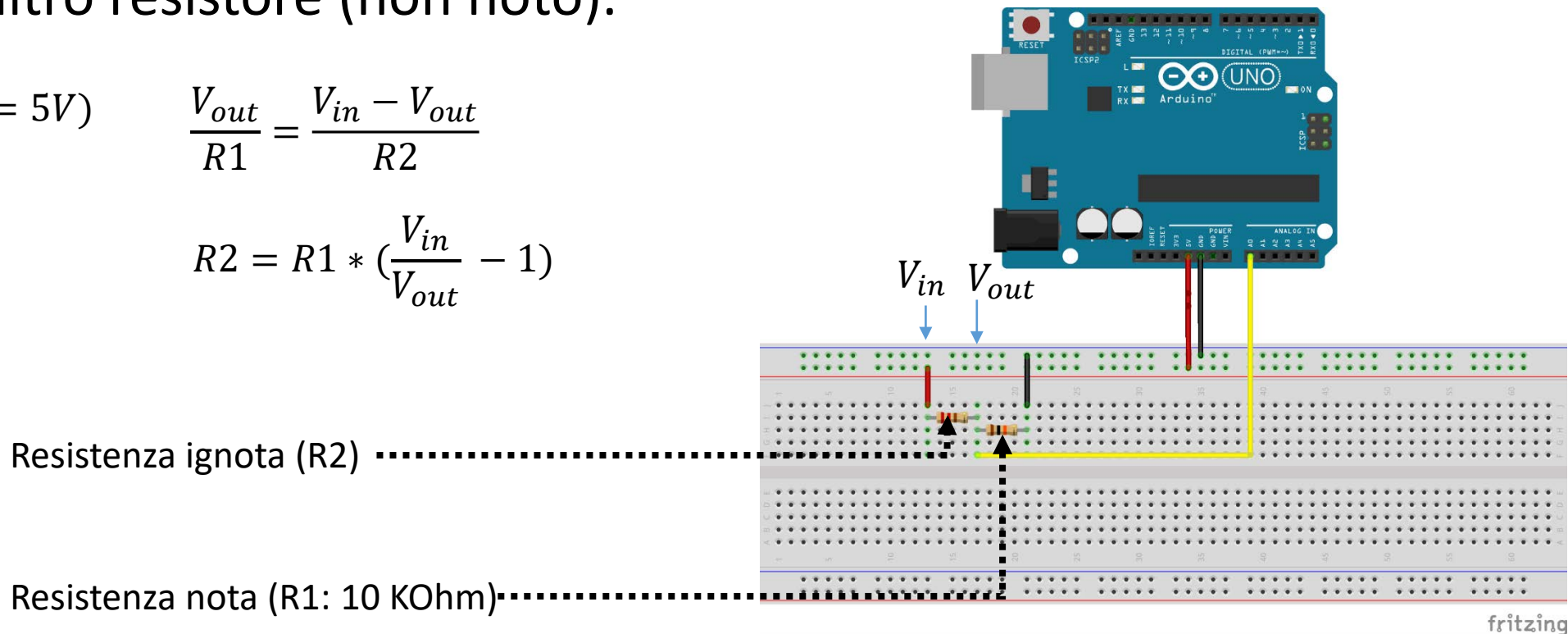
	Color	Significant figures			Multiply	Tolerance (%)	Temp. Coeff. (ppm/K)	Fail Rate (%)
Bad	black	0	0	0	x 1		250 (U)	
Beer	brown	1	1	1	x 10	1 (F)	100 (S)	1
Rots	red	2	2	2	x 100	2 (G)	50 (R)	0.1
Our	orange	3	3	3	x 1K		15 (P)	0.01
Young	yellow	4	4	4	x 10K		25 (Q)	0.001
Guts	green	5	5	5	x 100K	0.5 (D)	20 (Z)	
But	blue	6	6	6	x 1M	0.25 (C)	10 (Z)	
Vodka	violet	7	7	7	x 10M	0.1 (B)	5 (M)	
Goes	grey	8	8	8	x 100M	0.05 (A)	1(K)	
Well	white	9	9	9	x 1G			
Get	gold			3th digit only for 5 and 6 bands	x 0.1	5 (J)		
Some	silver				x 0.01	10 (K)		
Now!	none					20 (M)		



Progetto: misuratore di resistenze (I)

- A volte il codice colorato sui resistori può essere di difficile lettura.
- E' possibile usare un resistore di valore noto per misurare il valore di un altro resistore (non noto).

$$(V_{in} = 5V) \quad \frac{V_{out}}{R1} = \frac{V_{in} - V_{out}}{R2}$$
$$R2 = R1 * \left(\frac{V_{in}}{V_{out}} - 1 \right)$$



Progetto: misuratore di resistenze (II)

(sketch: Ohm_Meter.ino)

Codice:

```
#define READ_PIN A0 // Intermediate Reading PIN
#define V_IN 5      // Reference Voltage (5V)
#define R1 10000 // Known Res. Value (10 KOhm)
float v_out = 0.0;
int read_value = 0;
float r2 = 0.0;

void setup() {
    Serial.begin(9600);
}
```

```
void loop() {
    read_value = analogRead(READ_PIN);
    if(read_value) {
        v_out = read_value * V_IN/1024.0;
        r2= R1 * (V_IN/v_out -1);
        Serial.print("v_out: ");
        Serial.print(v_out);
        Serial.println(" V");
        Serial.print("r2: ");
        Serial.print(r2);
        Serial.println(" Ohm");
        delay(1000);
    }
}
```

Limitazione: se vi è troppa differenza fra R1 e R2, il calcolo non sarà molto preciso.

Misuratore di resistenze

The screenshot shows the Arduino IDE interface. The title bar reads "Ohm_Meter | Arduino 1.8.2". The menu bar includes "File", "Modifica", "Sketch", "Strumenti", and "Aiuto". The toolbar contains icons for saving, undo, redo, and other sketch functions. The main editor window shows the code for the "Ohm_Meter" sketch. The code defines constants for the reading pin, reference voltage, and known resistance, and implements a loop to read the voltage and calculate the resistance. The status bar at the bottom indicates "Caricamento completato" and shows memory usage: "Lo sketch usa 3662 byte (11%) dello spazio disponibile per i programmi" and "Le variabili globali usano 228 byte (11%) di memoria dinamica".

```

#define READ_PIN A0 // Intermediate Reading PIN
#define V_IN 5      // Reference Voltage (5V)
#define R1 1000    // Known Resistance Value (10 KOhm)

float v_out = 0.0;
int read_value = 0;
float r2 = 0.0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  read_value= analogRead(READ_PIN);
  if(read_value) {
    v_out = read_value * V_IN/1024.0;
    r2= R1 * (V_IN/v_out -1);
  }
}

```

Caricamento completato

Lo sketch usa 3662 byte (11%) dello spazio disponibile per i programmi.
Le variabili globali usano 228 byte (11%) di memoria dinamica.

3 Arduino/Genuino Uno su COM10

COM10 (Arduino/Genuino Uno)

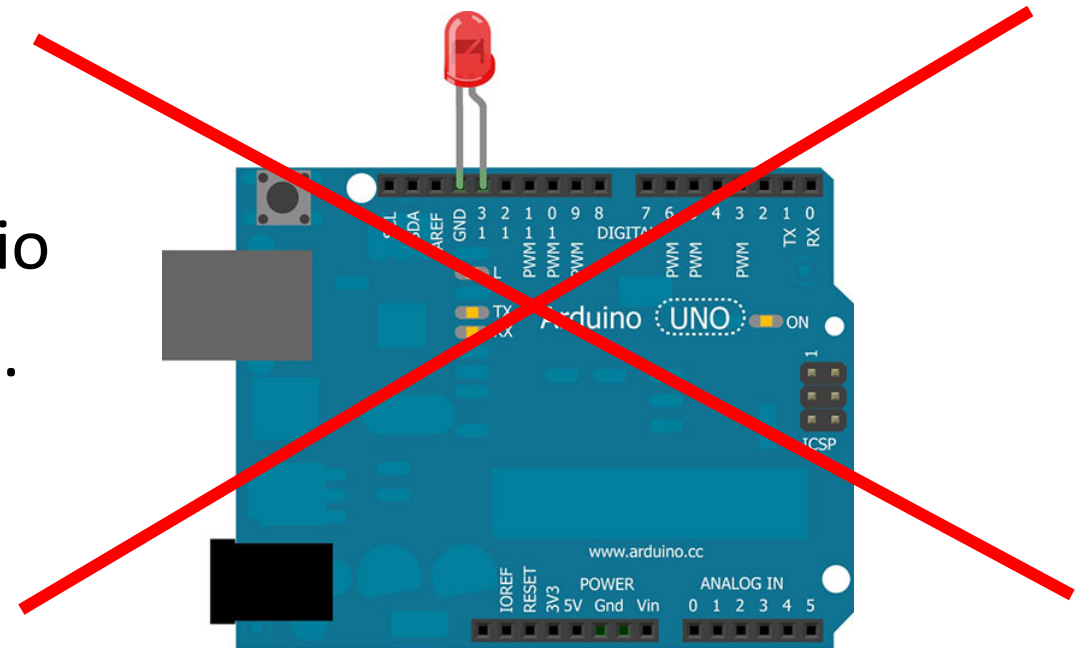
Invia

v_out: 0.14 V
 r2: 355714.31 Ohm
 v_out: 0.14 V
 r2: 355714.31 Ohm
 v_out: 0.14 V
 r2: 355714.31 Ohm
 v_out: 0.15 V
 r2: 331333.34 Ohm
 v_out: 0.14 V
 r2: 355714.31 Ohm
 v_out: 0.14 V
 r2: 355714.31 Ohm
 v_out: 0.14 V
 r2: 355714.31 Ohm

☒ Scorrimiento automatico Nessun fine riga 9600 baud

LED e resistori

- Nonostante in rete vi siano esempi che fanno uso di LED senza un resistore in serie, è meglio utilizzarne uno (per evitare danni al LED stesso o ai PIN di Arduino).
- Infatti senza il resistore la corrente che circola è più elevata (a parità di tensione).
- Se proprio si vuole correre il rischio, meglio scegliere un LED con voltaggio interno più elevato (LED verdi o blu).



Limiti operativi di Arduino

28.1 Absolute Maximum Ratings*

Dai documenti tecnici
Del chip ATmega328P.

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except <u>RESET</u> with respect to Ground	-0.5V to $V_{CC}+0.5V$
Voltage on <u>RESET</u> with respect to Ground.....	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins.....	200.0 mA

Nel caso del
collegamento di un
LED rosso:

LED Rosso: 1,8V
Resistenza: 220 Ohm
Intensità di corrente:
 $(5-1,8)/220=0,015$
 $A=15 \text{ mA} < 40\text{mA OK}$

Accendere un LED (sketch: LED_On.ino)

Codice:

```
#define LED_PIN 12

#define TIME_ON 7000

#define TIME_OFF 3000

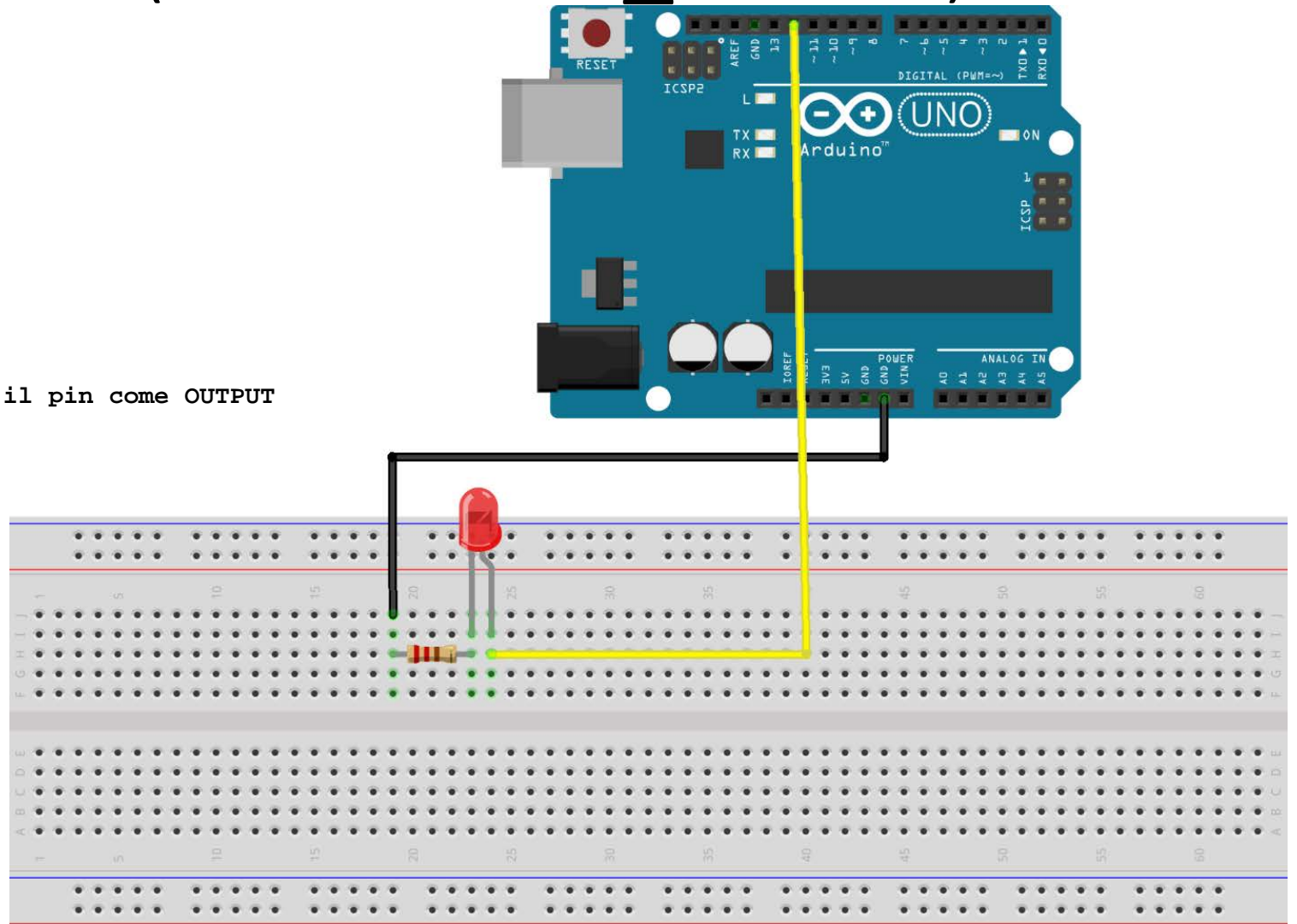
void setup() {
  pinMode(LED_PIN, OUTPUT); // Configuro il pin come OUTPUT
}

void loop() {
  // Accendo il LED
  digitalWrite(LED_PIN, HIGH);

  // Pausa (7 sec.)
  delay(TIME_ON);

  // Spengo il LED
  digitalWrite(LED_PIN, LOW);

  // Pausa (3 sec.)
  delay(TIME_OFF);
}
```



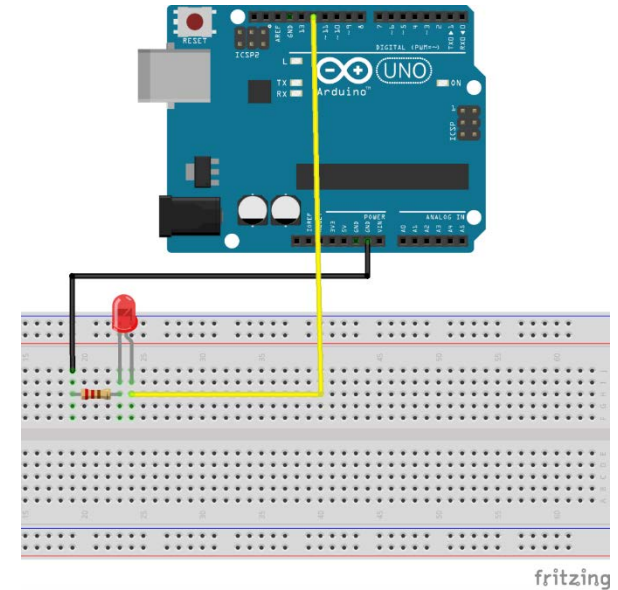
Far lampeggiare un LED (sketch: LED_Delay.ino)

Supponiamo di voler variare la luminosità del LED da minima a massima e viceversa, senza cambiare la resistenza usata nel circuito:

```
#define LED_PIN 12
#define PERIOD 30
#define STEP 2
int time_on, inc;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  time_on=0;
  inc=1;
}

void loop() {
  if(time_on>=0 && time_on<=PERIOD) {
    digitalWrite(LED_PIN, HIGH);
    delay(time_on);
    digitalWrite(LED_PIN, LOW);
    delay(PERIOD-time_on);
  }
  if(inc) {
    if(time_on<PERIOD) { time_on+=STEP; }
    else { inc=0; time_on=PERIOD; }
  }
  else {
    if(time_on>0) { time_on-=STEP; }
    else { inc=1; time_on=0; }
  }
}
```



Cosa succede aumentando il valore di PERIOD?

Pulse Width Modulation (PWM)

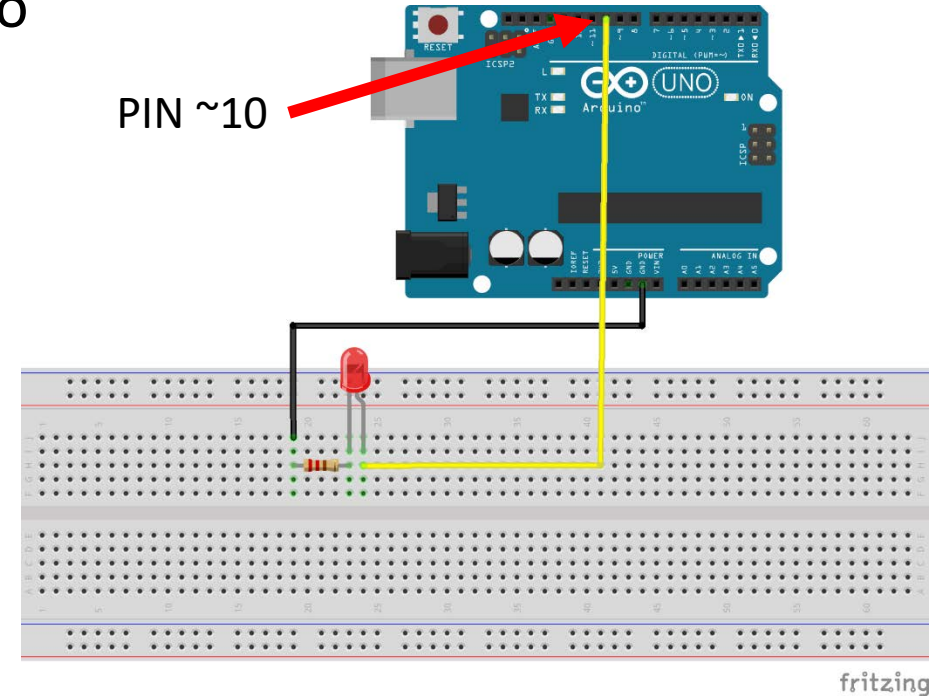
modulazione di larghezza di impulso (sketch: LED_Fade.ino)

Possiamo variare la luminosità del LED anche usando la funzione `analogWrite()` su un PIN che supporti la PWM (PIN il cui numero è preceduto da una ~):

```
#define LED_PIN 10
#define PAUSE_TIME 30

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

void loop() {
  for(int i=0; i<=255; i+=5) {
    analogWrite(LED_PIN, i);
    delay(PAUSE_TIME);
  }
  for(int i=255; i>=0; i-=5) {
    analogWrite(LED_PIN, i);
    delay(PAUSE_TIME);
  }
}
```



Buzzer (Piezo Speaker)

- Un buzzer è un componente che sfrutta la “piezoelettricità” per emettere dei suoni.
- Infatti contiene un cristallo che cambia forma quando viene applicata ad esso dell'elettricità. Applicando il segnale elettrico con la giusta frequenza il cristallo può emettere dei suoni.



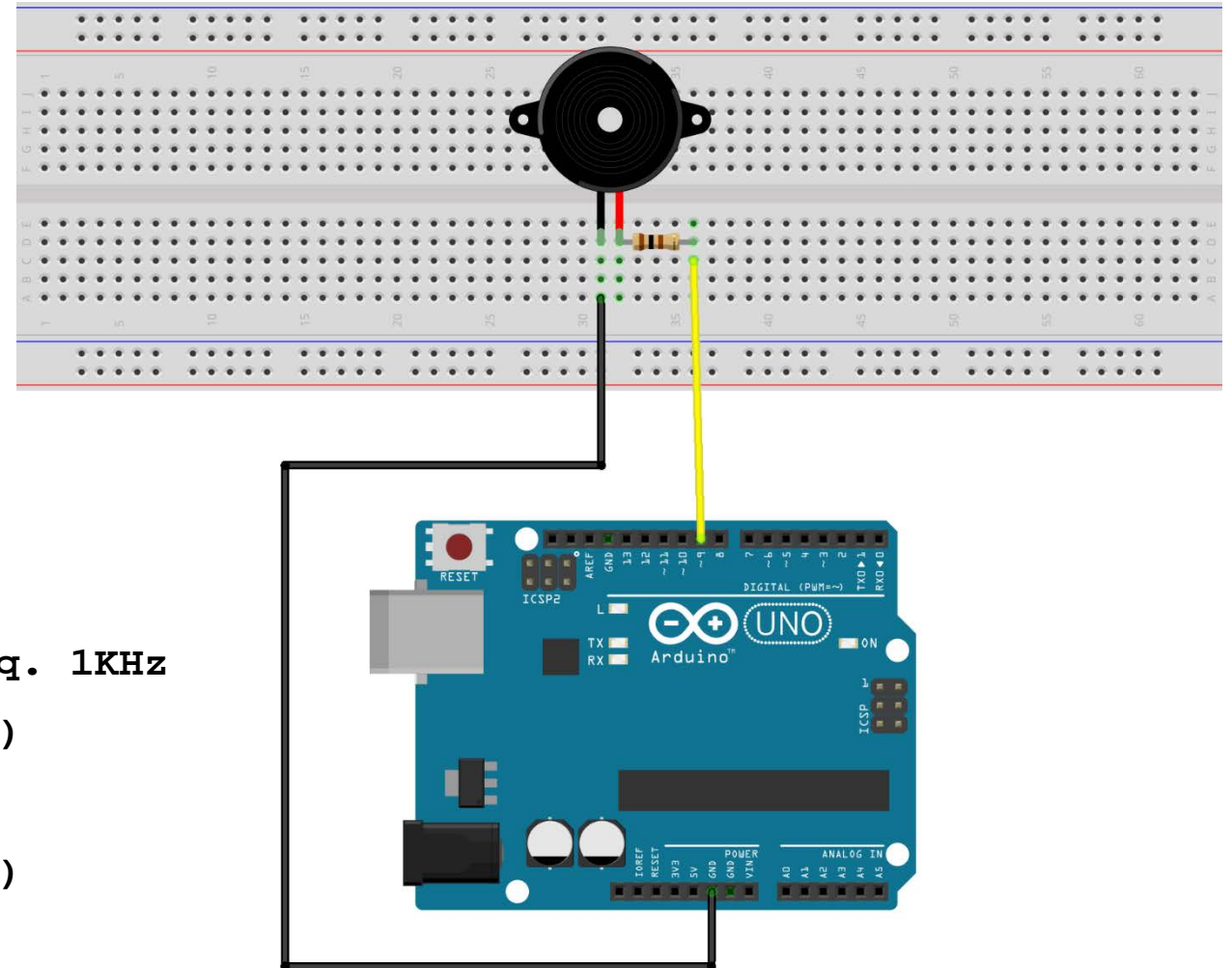
Emettere suoni con un buzzer (sketch: Buzzer.ino)

Codice:

```
#define BUZZ_PIN 9

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

void loop(){
  tone(BUZZ_PIN, 1000); // Suono con freq. 1KHz
  delay(1000);           // Pausa (1 sec.)
  noTone(BUZZ_PIN);      // Stop
  delay(1000);           // Pausa (1 sec.)
}
```



Fotoresistore

- Il fotoresistore è un componente la cui resistenza è inversamente proporzionale alla quantità di luce che lo colpisce.



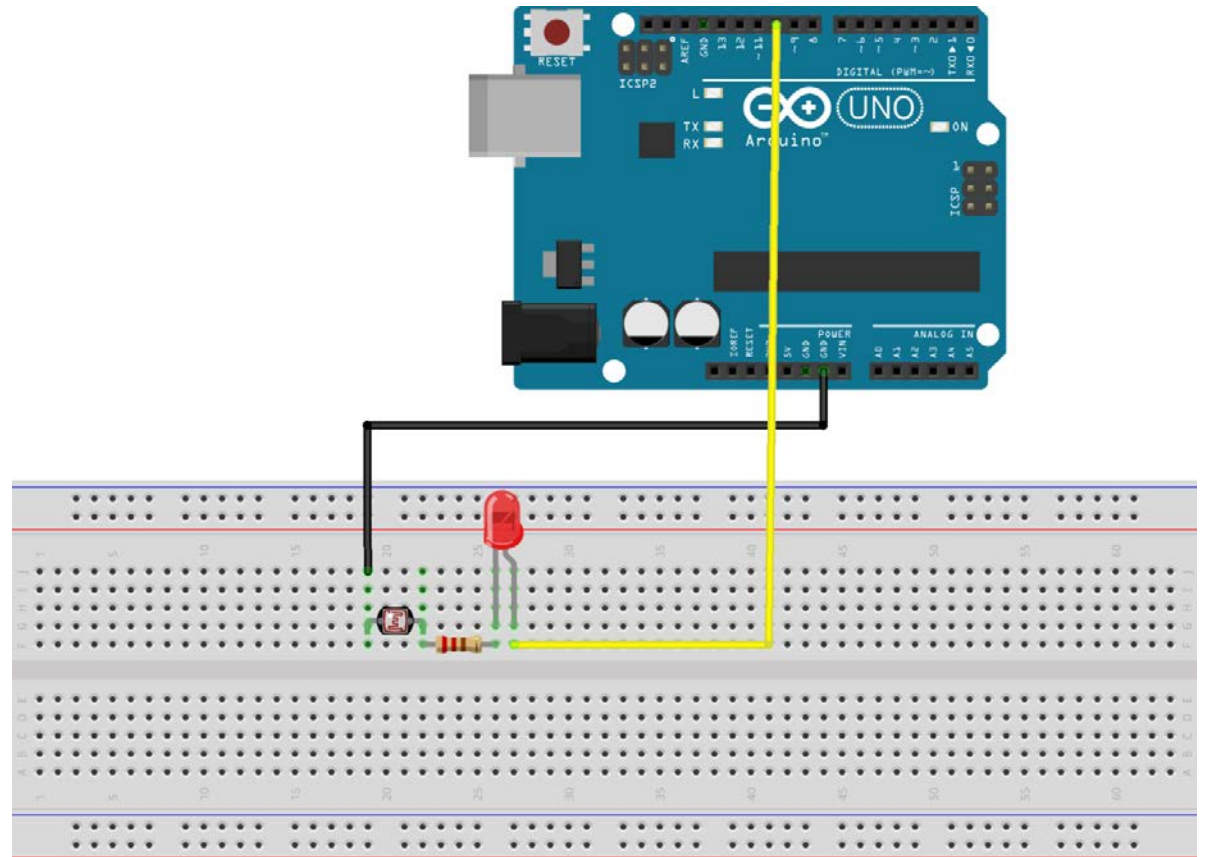
Esempio (sketch: LED_Photoristor.ino)

- Facendo ombra sul fotoresistore, la luce del LED si affievolisce fino a spegnersi (quasi) del tutto.

```
#define LED_PIN 12
```

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

```
void loop() {  
  // Accendo il LED  
  digitalWrite(LED_PIN, HIGH);  
}
```



Pulsante

- Dispositivo che converte la pressione esercitata su di esso in segnale elettrico.



Esempio (sketch: Button.ino)

- Usiamo un pulsante per accendere un LED:

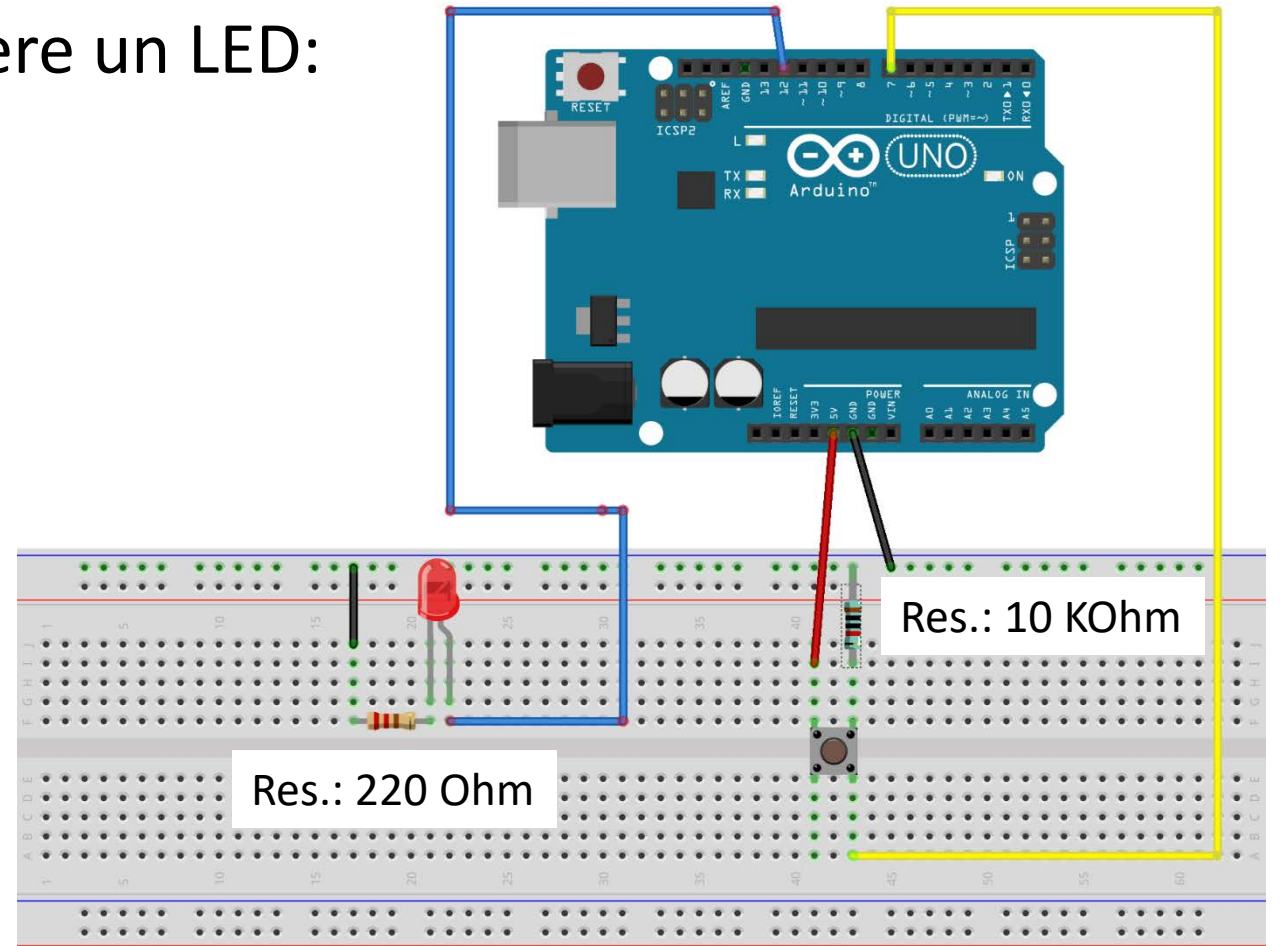
```
#define BUTTON_PIN 7
#define LED_PIN 12

int buttonState = 0;

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

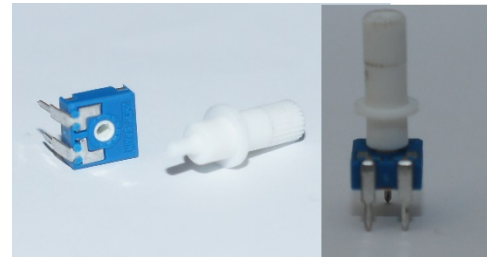
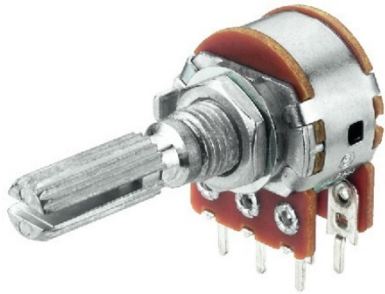
void loop() {
  buttonState = digitalRead(BUTTON_PIN);

  if (buttonState == HIGH) {
    digitalWrite(LED_PIN, HIGH);
  } else {
    digitalWrite(LED_PIN, LOW);
  }
}
```



Potenziometro

- Un potenziometro è un partitore di tensione regolabile.
- Può essere utilizzato come un reostato, ovvero, permette di ottenere una resistenza variabile.



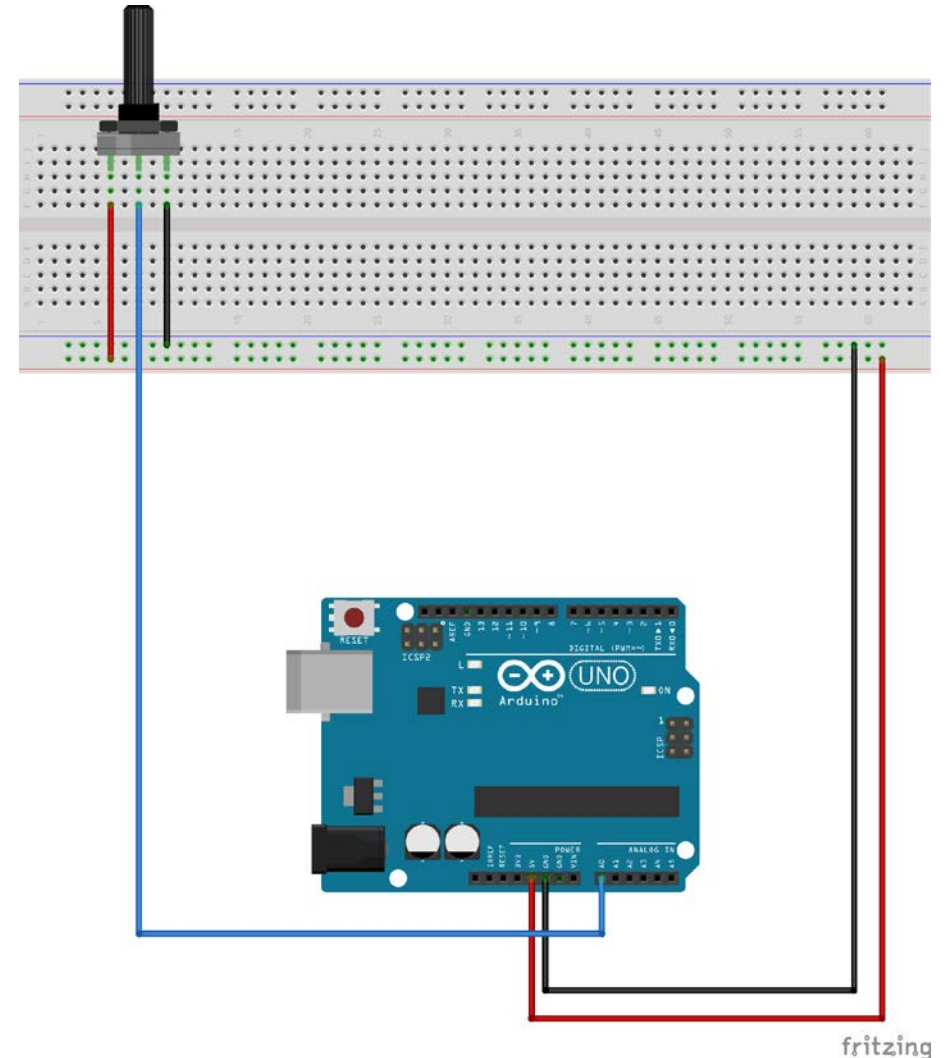
Arduino IDE: monitorare un potenziometro (sketch: Potentiometer.ino)

Codice:

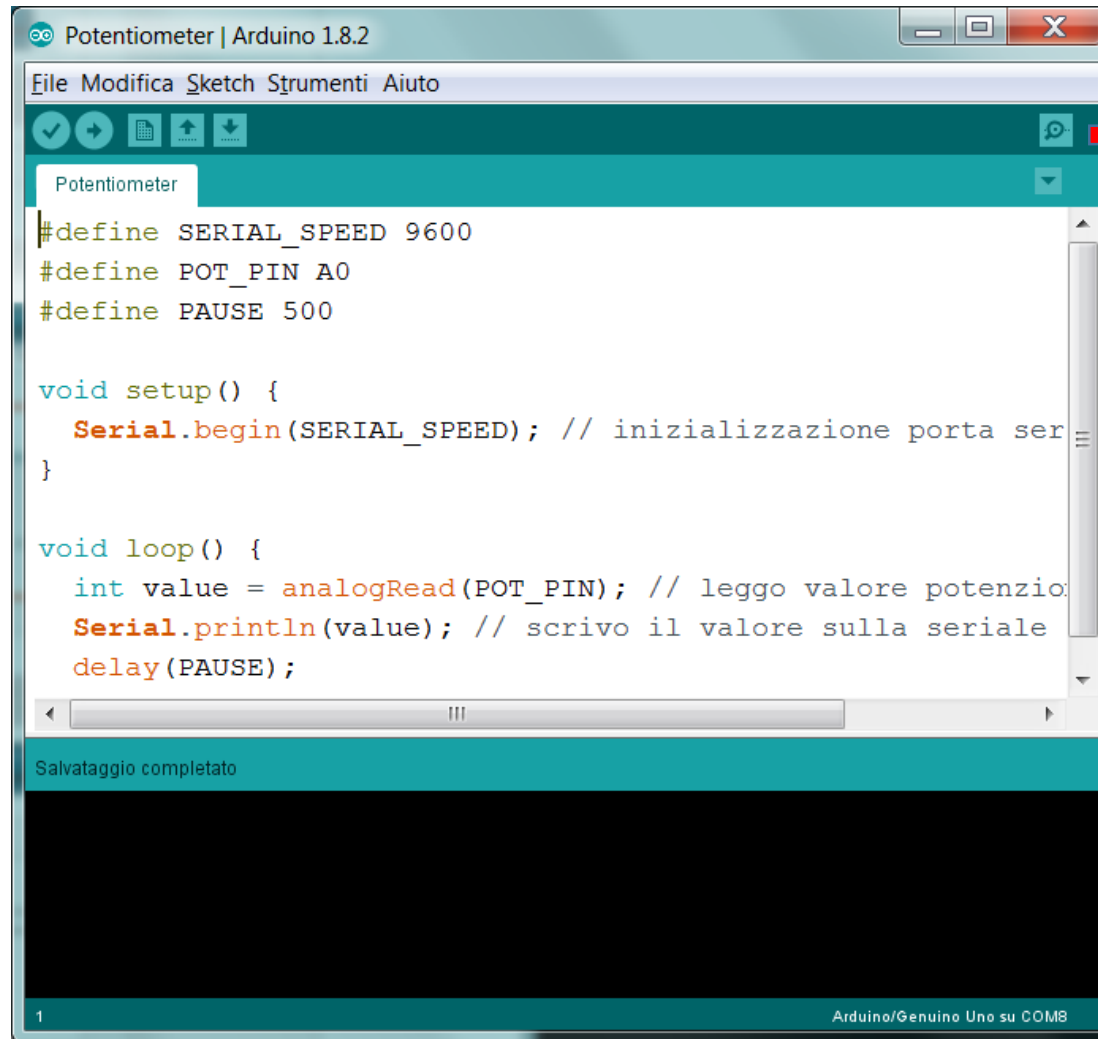
```
#define SERIAL_SPEED 9600
#define POT_PIN A0
#define PAUSE 500

void setup() {
  Serial.begin(SERIAL_SPEED); // inizializzazione porta seriale
}

void loop() {
  int value = analogRead(POT_PIN); // leggo valore potenziometro
  Serial.println(value); // scrivo il valore sulla seriale
  delay(PAUSE);
}
```



Arduino IDE: monitorare un potenziometro



Potentiometer | Arduino 1.8.2

File Modifica Sketch Strumenti Aiuto

Potentiometer

```
#define SERIAL_SPEED 9600
#define POT_PIN A0
#define PAUSE 500

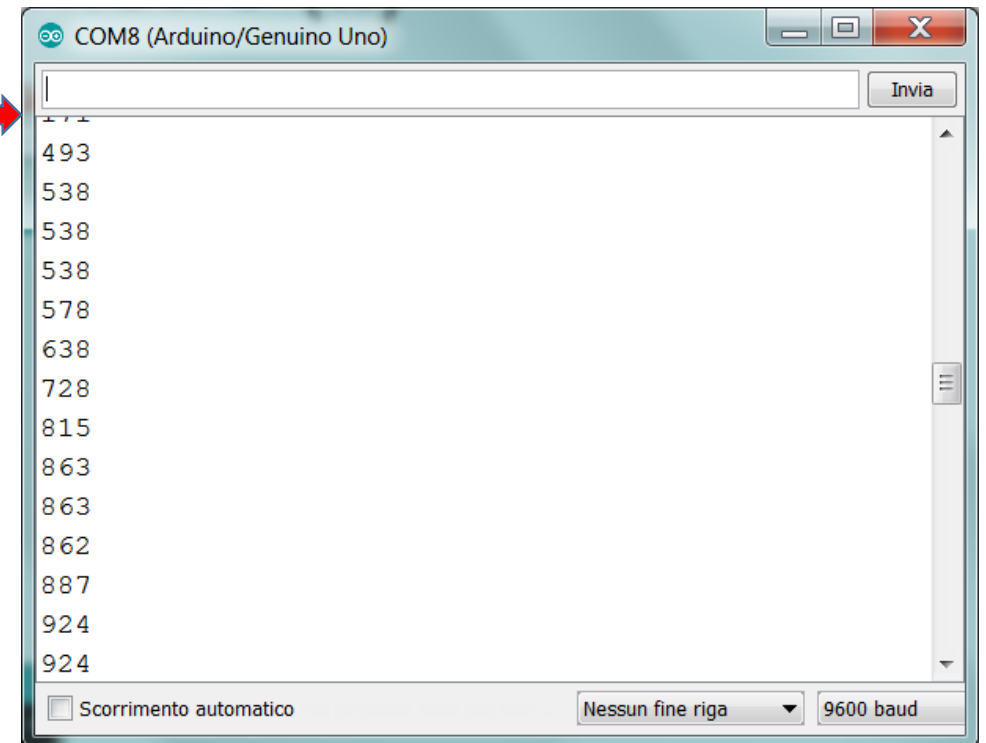
void setup() {
  Serial.begin(SERIAL_SPEED); // inizializzazione porta ser
}

void loop() {
  int value = analogRead(POT_PIN); // leggo valore potenziometro
  Serial.println(value); // scrivo il valore sulla seriale
  delay(PAUSE);
}
```

Salvataggio completato

1

Arduino/Genuino Uno su COM8



COM8 (Arduino/Genuino Uno)

Invia

493
538
538
538
578
638
728
815
863
863
862
887
924
924

☒ Scorrimento automatico

Nessun fine riga

9600 baud

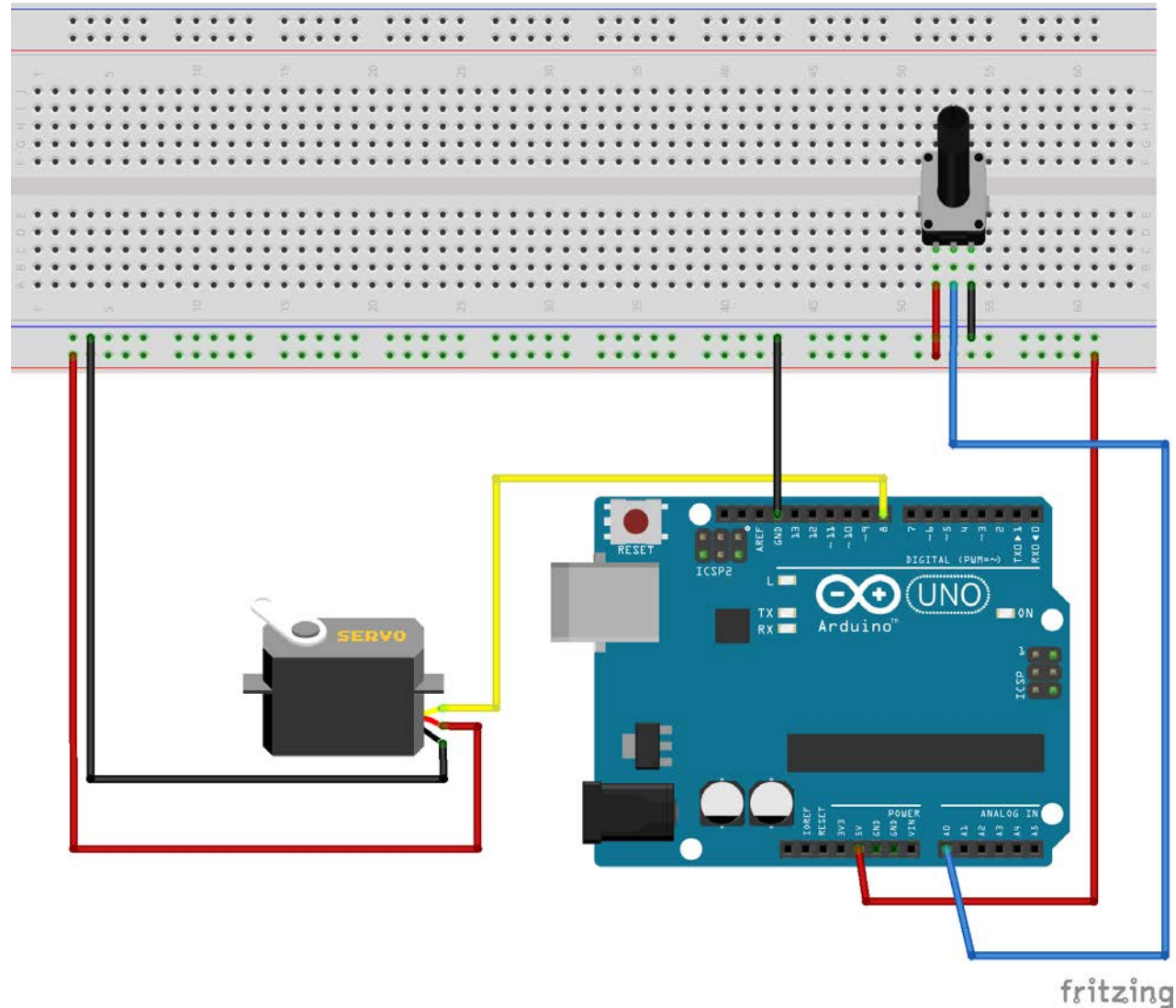
Servomotore

- Si tratta di un tipo di motore che può ruotare di 180 gradi: gli impulsi ricevuti dicono al motore quale posizione assumere.



Servomotore: schema del circuito

Idea: servirsi di un potenziometro per comandare il servomotore.



Servomotore: codice (sketch Servomotor.ino)

```
#include <Servo.h>
#define POT_PIN A0                // PIN Analogico del Potenzziometro
#define MOTOR_PIN 8               // PIN (digitale) del Servomotore
#define SERIAL_SPEED 9600

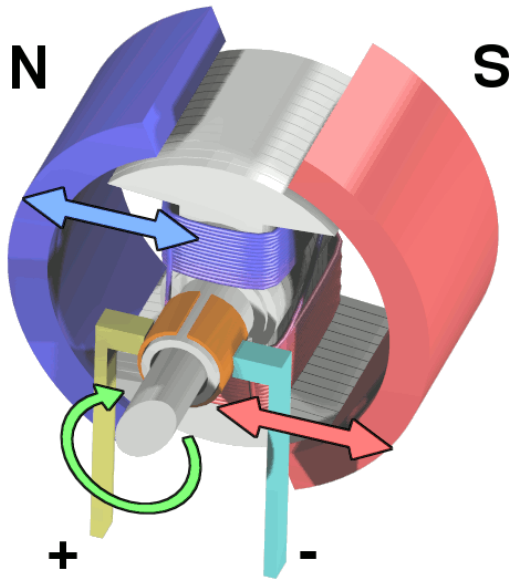
Servo motor;

void setup()
{
    motor.attach(MOTOR_PIN);
    Serial.begin(SERIAL_SPEED);
    delay(1000);
}
void loop()
{
    int val = analogRead(POT_PIN);
    val = map(val, 0, 1023, 0, 179); // mappa il valore letto dal pot.
                                     // nell'intervallo 0-179 gradi

    Serial.println(val);
    motor.write(val);
}
```

Motore a corrente continua (DC Motor 6/9V)

- Un motore a corrente continua converte energia elettrica in energia meccanica.



Quando la corrente scorre negli avvolgimenti, si genera un campo magnetico intorno al rotore. La parte sinistra del rotore è respinta dal magnete di sinistra ed attratta da quello di destra. Analogamente fa la parte in basso a destra. La coppia genera la rotazione. (Fonte: Wikipedia)

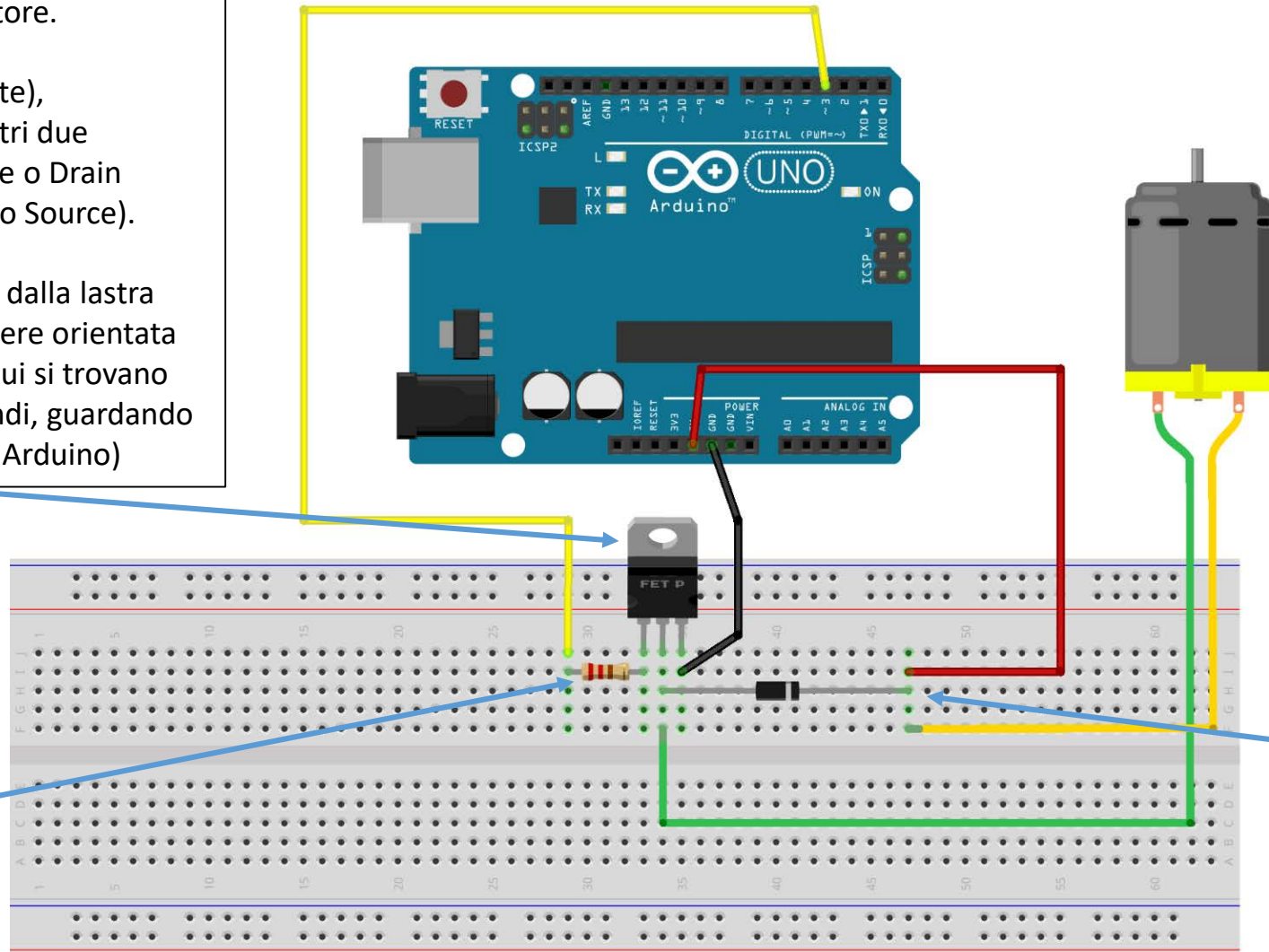


DC Motor: schema del circuito.

Transistor: funge da interruttore. Quando arriva della corrente sul primo piedino (Base o Gate), si chiude il contatto tra gli altri due (il secondo è detto Elettrodo o Drain ed il terzo è detto Collettore o Source).

Attenzione: la parte rivestita dalla lastra metallica con il foro deve essere orientata dal lato opposto a quello in cui si trovano la resistenza ed il diodo (quindi, guardando il disegno, verso la scheda di Arduino)

Resistenza: 220 Ohm



Diodo: fa in modo che la corrente fluisca solo in un senso.

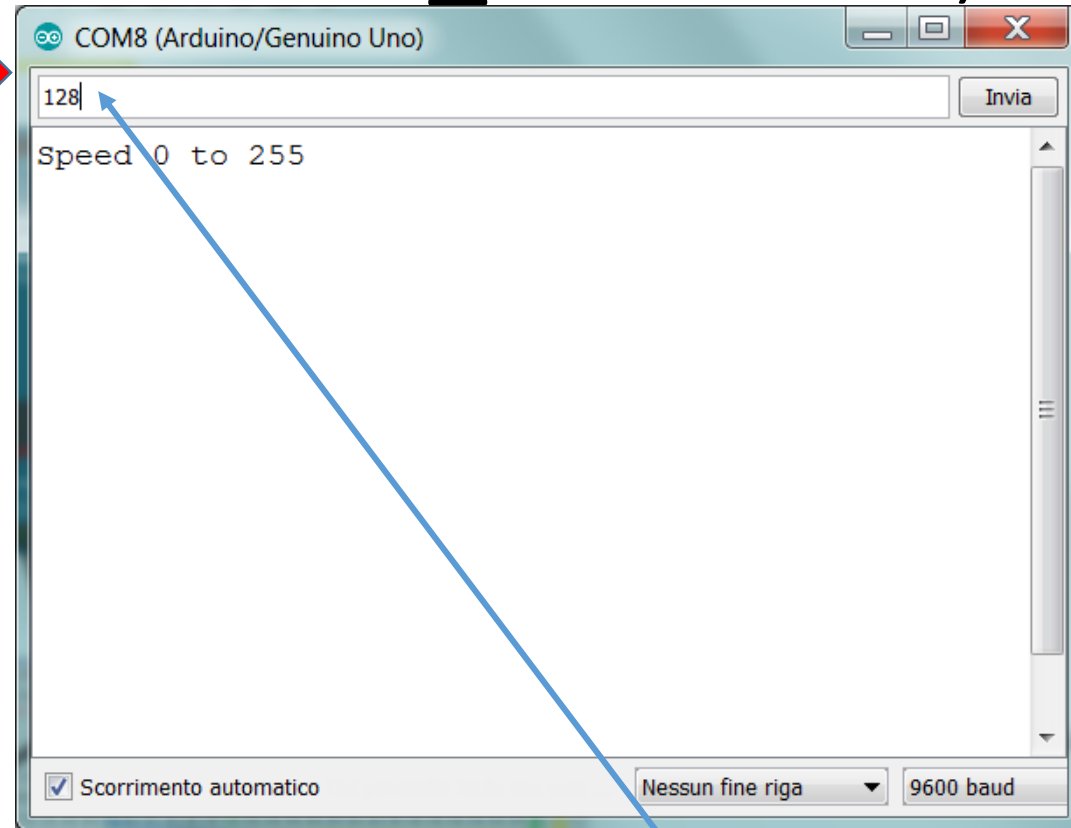
Attenzione: la parte con la fascia argentata deve essere rivolta verso i cavi collegati al PIN 5V di Arduino ed al polo positivo del motore.

DC Motor: codice (sketch: DC_Motor.ino)

```
File Modifica Sketch Strumenti Aiuto
DC_Motor
int motorPin = 3;

void setup()
{
  pinMode(motorPin, OUTPUT);
  Serial.begin(9600);
  while (! Serial);
  Serial.println("Speed 0 to 255");
}

void loop()
{
  if (Serial.available())
  {
    int speed = Serial.parseInt();
    if (speed >= 0 && speed <= 255)
    {
      analogWrite(motorPin, speed);
    }
    else
      digitalWrite(motorPin, LOW);
  }
}
```



Inserire un valore fra 0 (spento) e 255 (velocità massima) e premere invio per far ruotare il motore ad una certa velocità

DC Motor: variante per Linux

(sketch: DC_Motor_Linux)

```
int motorPin = 3;
int lastSpeed=0;

void setup()
{
  pinMode(motorPin, OUTPUT);
  Serial.begin(9600);
  while (! Serial);
  Serial.println("Speed 0 to 255");
}

void loop() {
  if (Serial.available()) {
    int speed = Serial.parseInt();

    if (speed > 0 && speed <= 255) {
      lastSpeed=speed;
    }

    if (speed==-1)
      lastSpeed=0;
  }

  analogWrite(motorPin, lastSpeed);
}
```

- In Linux può capitare che, dopo aver inserito una velocità non nulla con il Serial Monitor, **il motore si fermi dopo poche rotazioni.**
- Ciò avviene perché, nel monitor della porta seriale, di default, è selezionato un «fine riga» (e.g., newline, CR o entrambi).
- In mancanza di un continuo inserimento da parte dell'utente, il fine riga (convertito in **0**) ferma il motore.
- In questa variante per arrestare il motore bisogna inserire **-1**.
- La variante va bene anche per Windows se si abilita il fine riga.

Parte II

Arduino e Processing

Preparazione di Processing e di Arduino

- Installazione della libreria **Arduino (Firmata)** in Processing:
 - Menu *Sketch / Importa Libreria... / Manage Libraries...*
 - Cercare e selezionare **Arduino (Firmata)** di David A. Mellis.
 - Oppure scaricare e installare la libreria manualmente da <https://github.com/firmata/processing/releases/tag/latest>
 - Documentazione:
 - <https://playground.arduino.cc/Interfacing/Processing>
 - Scaricare l'**Arduino IDE** da <https://www.arduino.cc/en/Main/Software>
 - Collegare la board Arduino via USB al PC
 - Lanciare **Arduino IDE** e selezionare
 - dal menu *Strumenti / Scheda* selezionare il tipo di scheda (Arduino/Genuino Uno)
 - dal menu *Strumenti / Porta* selezionare la porta a cui è collegata la scheda
 - dai menu *File / Esempi / Firmata / Standard Firmata*
 - Caricare il programma **Standard Firmata** sulla board Arduino (*Sketch / Carica*, o Ctrl+U o pulsante freccia a destra).

Processing API: un'occhiata all'interfaccia di programmazione

- **Arduino.list()**: lista di dispositivi seriali disponibili sul PC (se Arduino è collegato sarà uno di questi).
- **Arduino(parent, name, rate)**: crea un oggetto di tipo Arduino (parent assumerà il valore this, name sarà il nome della porta seriale corrispondente e rate sarà la velocità della connessione, e.g., 57600).
- **pinMode(pin, mode)**: imposta un digital pin come input, output, o servo mode (Arduino.INPUT, Arduino.OUTPUT o Arduino.SERVO).
- **digitalRead(pin)**: restituisce il valore di un digital pin, ovvero, Arduino.LOW o Arduino.HIGH (il pin deve essere impostato come input).
- **digitalWrite(pin, value)**: invia Arduino.LOW o Arduino.HIGH ad un digital pin.
- **analogRead(pin)**: restituisce il valore di un analog input (da 0 a 1023).
- **analogWrite(pin, value)**: invia un valore analogico (PWM wave) ad un digital pin che supporta questa modalità (pin 3, 5, 6, 9, 10 e 11); il valore può variare da 0 (sempre off) a 255 (sempre on).
- **servoWrite(pin, value)**: invia un valore ad un servo motore; il valore può variare da 0 a 180.

Un semplice test (sketch: OnBoardLED.pde).

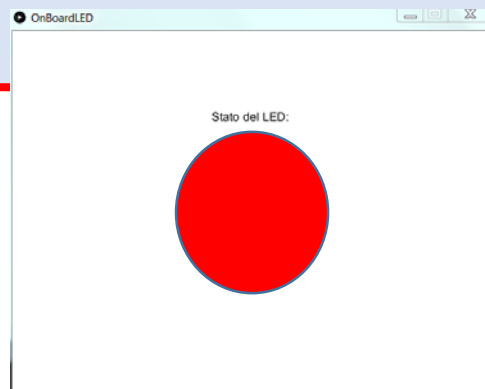
- Accendiamo il LED sulla scheda di Arduino (13) cliccando il tasto sinistro del mouse.
- Contemporaneamente impostiamo a "rosso" il colore di un cerchio centrato nella finestra:

```
...  
try {  
    arduino=new Arduino(this, COM_PORT, 57600);  
}  
catch(Exception e) {  
    println("Errore: "+e.toString());  
}  
...
```

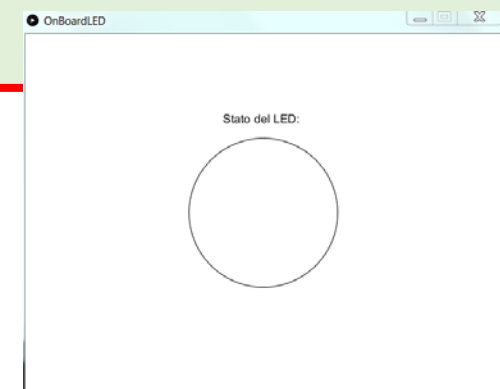
Attenzione: in linea 9 sostituire "COM8" con il nome della porta a cui è connesso Arduino:

static final String
COM_PORT="COM8";

```
void mousePressed() {  
    if(arduino!=null) {  
        arduino.pinMode(LED_PIN, Arduino.OUTPUT);  
        arduino.digitalWrite(LED_PIN, Arduino.HIGH);  
        ellipseColor=#FF0000;  
        println("LED "+LED_PIN+" acceso");  
    }  
}
```



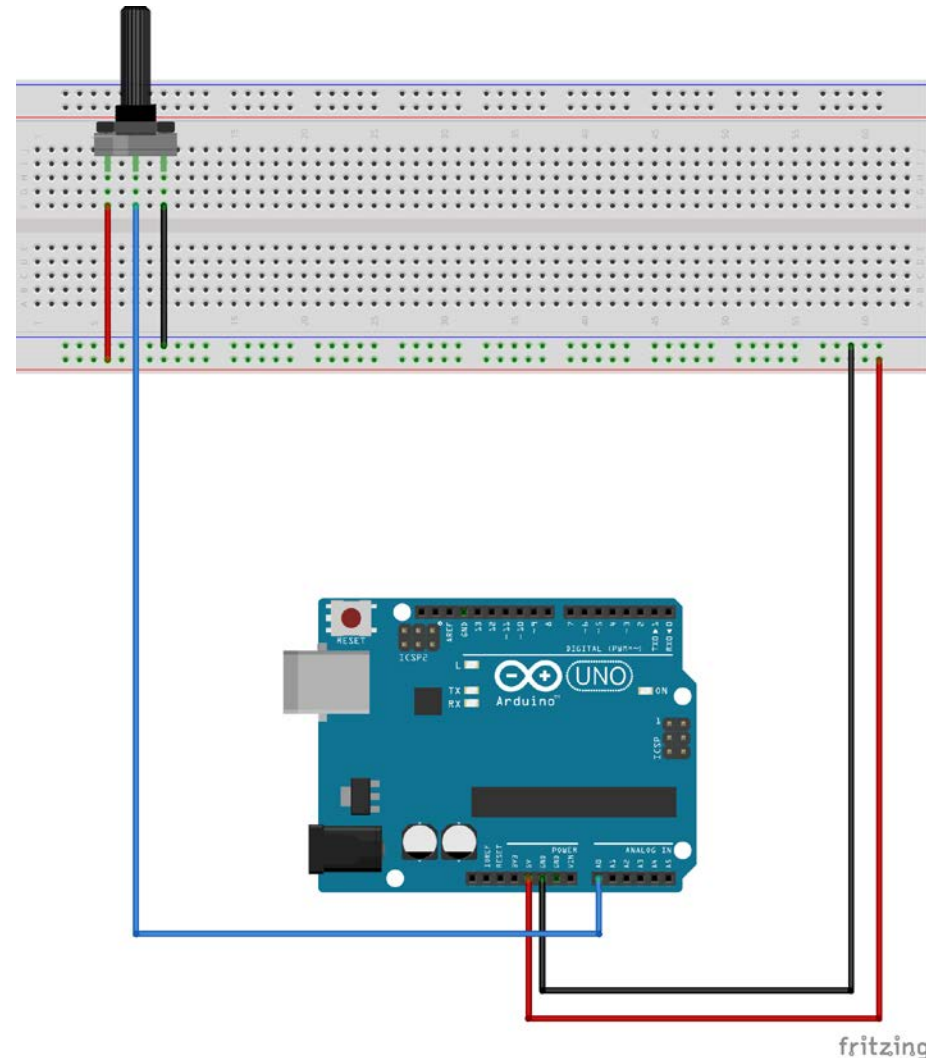
```
void mouseReleased() {  
    if(arduino!=null) {  
        arduino.pinMode(LED_PIN, Arduino.OUTPUT);  
        arduino.digitalWrite(LED_PIN, Arduino.LOW);  
        ellipseColor=#FFFFFF;  
        println("LED "+LED_PIN+" spento");  
    }  
}
```



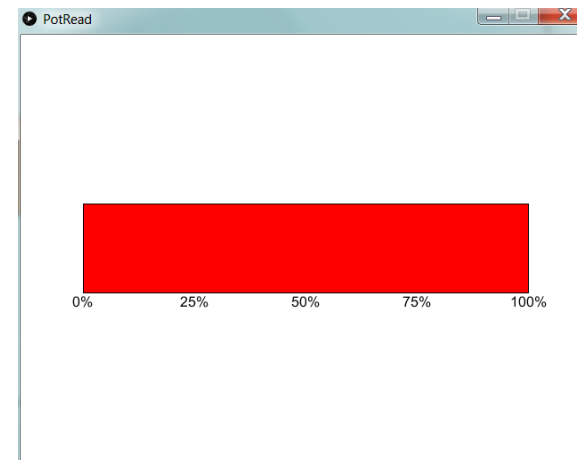
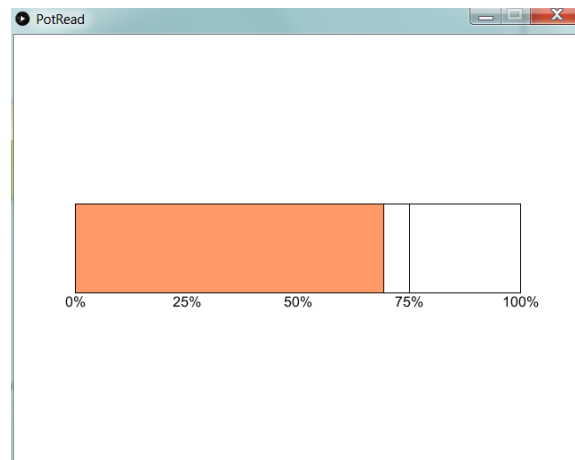
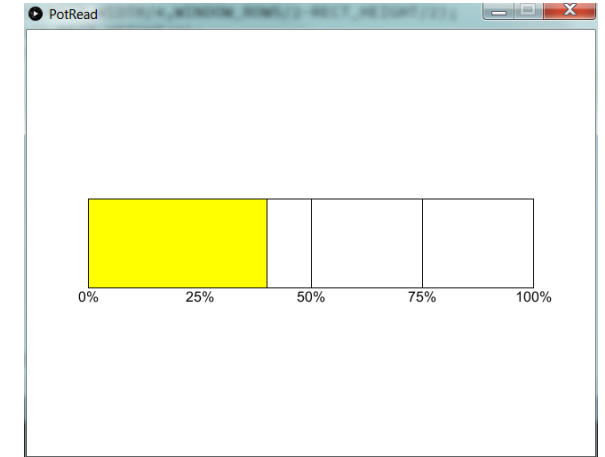
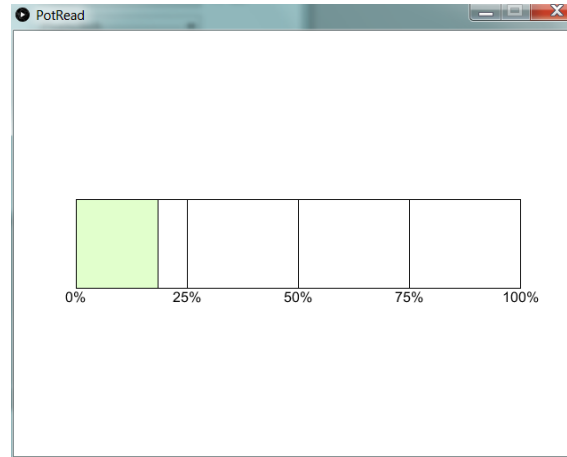
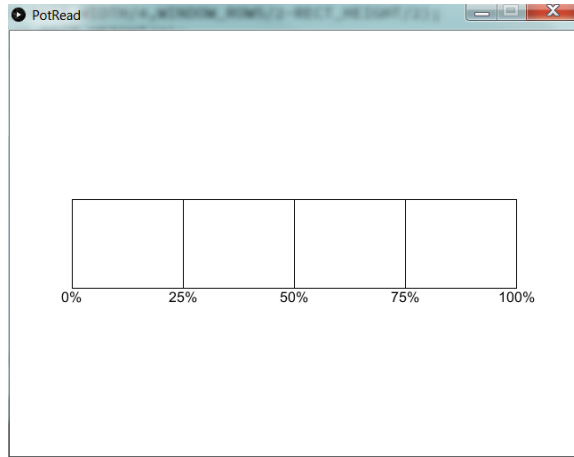
Processing: monitorare un potenziometro (sketch: PotRead.pde)

- Idea: rappresentare il valore assunto da un potenziometro mediante un rettangolo che si riempie di un colore diverso a seconda del livello raggiunto dal segnale.
- **Attenzione**: ricordarsi di caricare il programma StandardFirmata dall'Arduino IDE, prima di eseguire lo sketch Processing.
- **Attenzione**: in linea 9 sostituire "COM8" con il nome della porta a cui è connesso Arduino:

`static final String COM_PORT="COM8";`



Processing: monitorare un potenziometro



Processing: monitorare un potenziometro

- Uno sguardo al codice che disegna l'area colorata:

```
pg.rectMode(CENTER);  
...  
int val=arduino.analogRead(potpin); // Legge il valore del sensore (un intero fra 0 e 1023)  
int mappedVal=(int)map(val,0,1023,0,RECT_WIDTH); // Mappa il valore sulla lunghezza del rett.  
println("*****"+mappedVal+"*****");  
  
if(mappedVal<=RECT_WIDTH/4) // Valore inferiore o uguale al 25%  
    pg.fill(225,255,204);      // Verde chiaro  
if(mappedVal>RECT_WIDTH/4 && mappedVal<=RECT_WIDTH/2) // Valore compreso fra 25% e 50%  
    pg.fill(255,255,0);        // Giallo  
if(mappedVal>=RECT_WIDTH/2 && mappedVal<=RECT_WIDTH/4*3) // Valore fra il 50% ed il 75%  
    pg.fill(255,153,102);      // Arancione  
if(mappedVal>RECT_WIDTH/4*3) // Valore maggiore al 75%  
    pg.fill(255,0,0);          // Rosso  
pg.rect(WINDOW_COLS/2-RECT_WIDTH/2+mappedVal/2,WINDOW_ROWS/2,mappedVal,RECT_HEIGHT);
```

Parte III

Progetti avanzati

Progetto 1: usare un sensore PIR

- Usiamo un sensore PIR (Passive InfraRed) per realizzare un rilevatore di movimento.
- Un sensore PIR misura i raggi infrarossi irradiati nel suo campo d'azione.



- Accenderemo un led quando il sensore rileverà un oggetto.

Schema del progetto

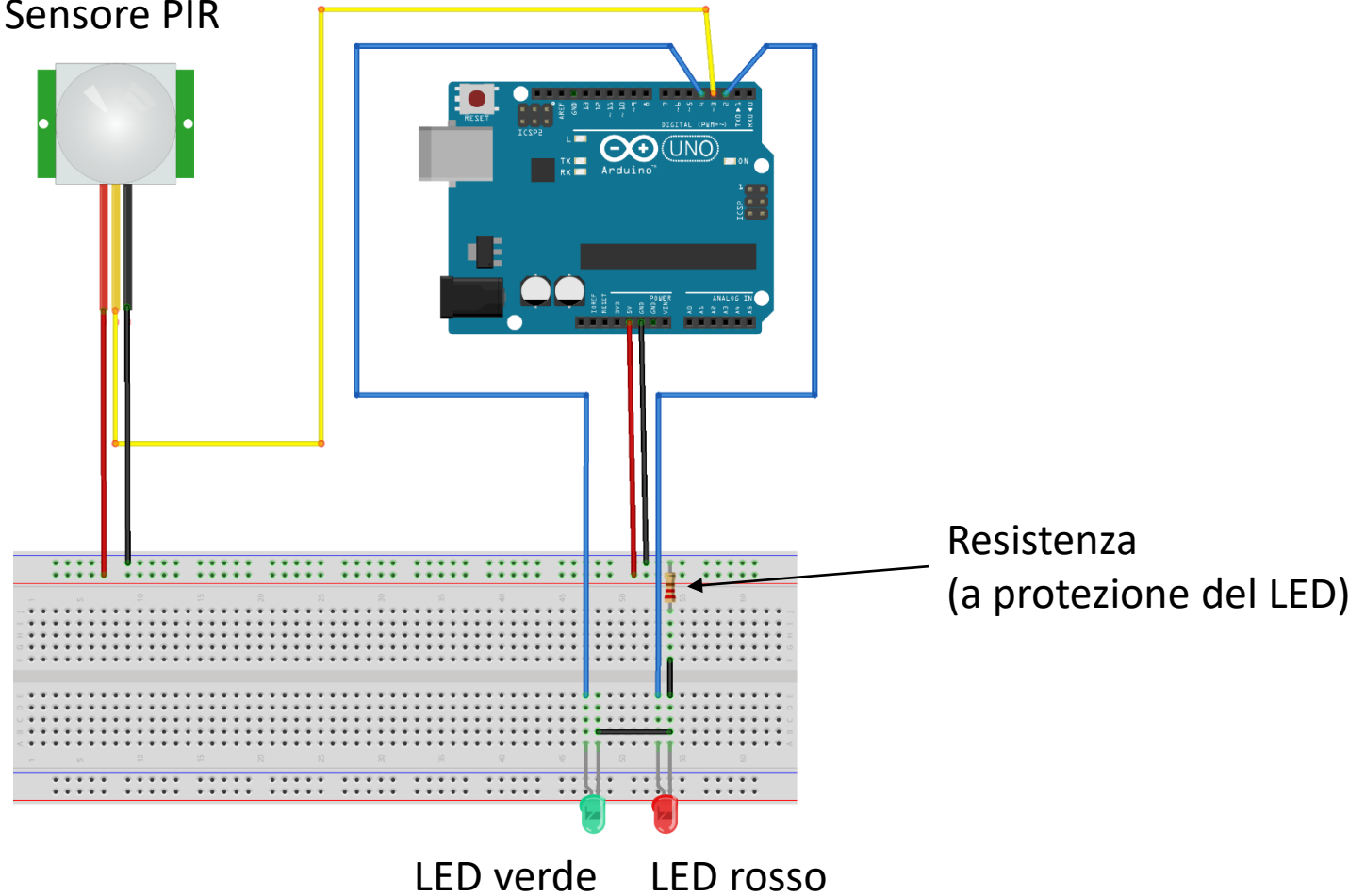
Colore cavi:

Rosso: tensione (+5V)

Nero: terra (ground)

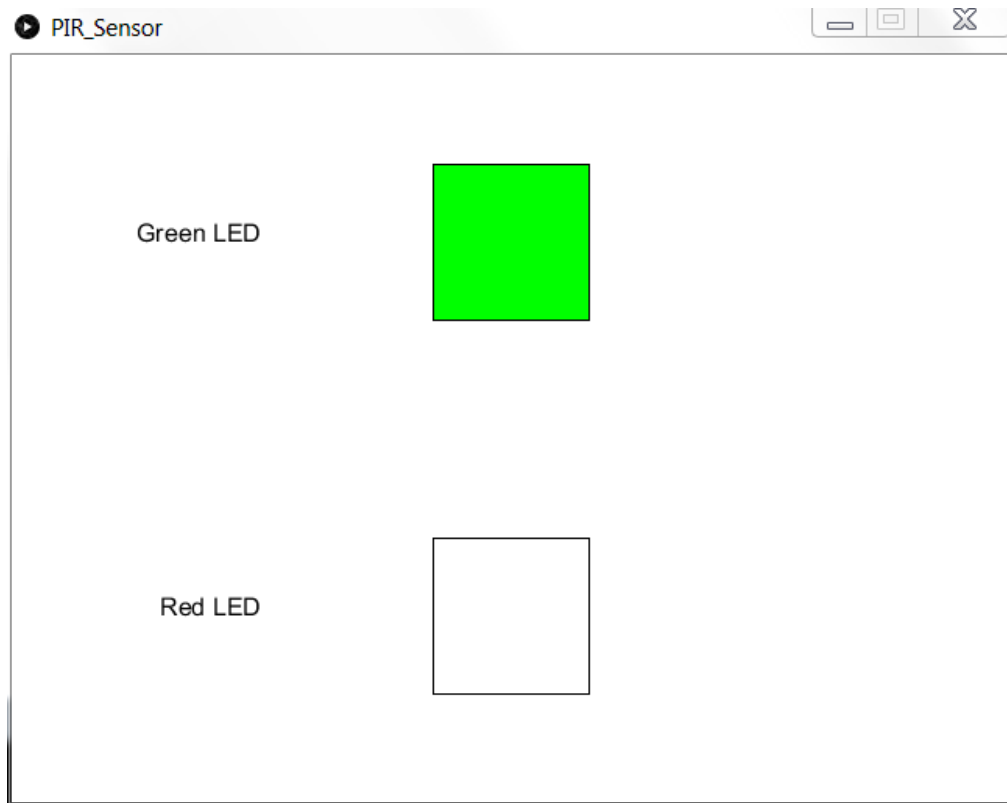
Blu/giallo: segnale

Sensore PIR

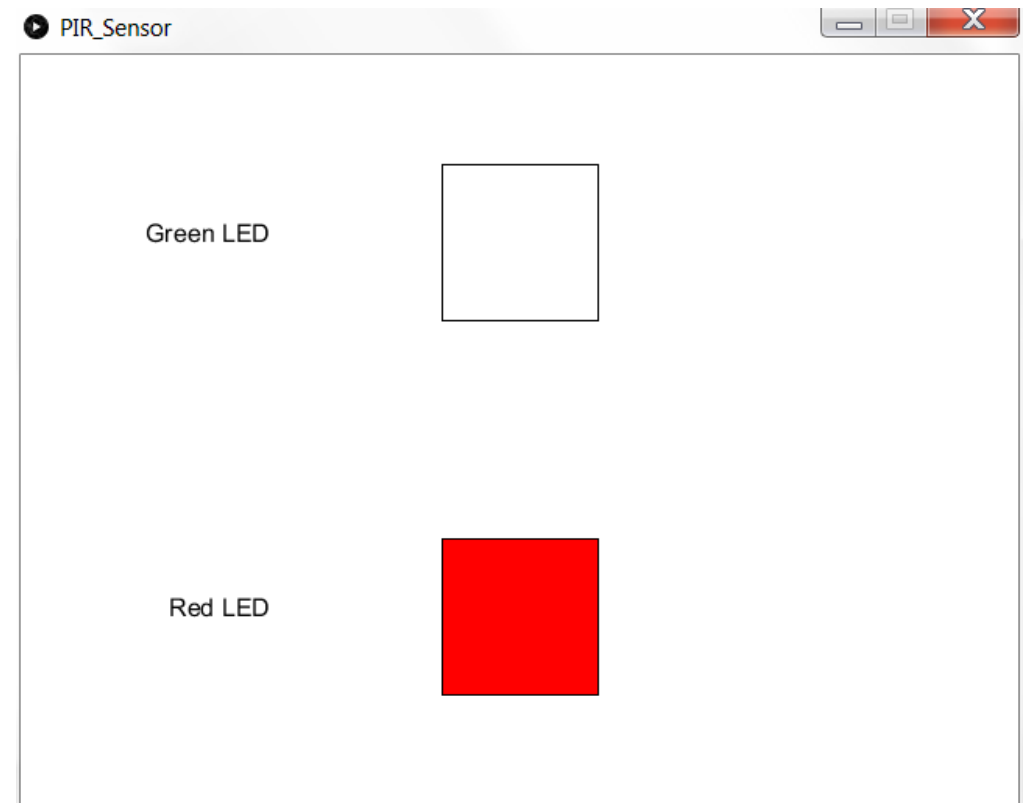


Funzionamento

Fase 1: LED verde acceso, nessun movimento rilevato



Fase 2: LED rosso acceso, movimento rilevato



Uno sguardo al codice (I)

```
import processing.serial.*; ← Importiamo le librerie necessarie
import cc.arduino.*;
...
int pirPin = 3; // PIN collegato al sensore PIR
int greenLedPin = 4; // PIN collegato al LED verde
int redLedPin = 2; // PIN collegato al LED rosso
Arduino arduino; // oggetto di tipo Arduino: serve a
dialogare con la board
boolean arduinoCheck = false; // booleano: true ->
Arduino collegato, false -> Arduino assente
```

Uno sguardo al codice (II)

```
void setup() {  
  ...  
  try {  
    println((Object[])Arduino.list());  
    arduino = new Arduino(this, Arduino.list()[2], 57600); // inizializza l'oggetto arduino  
    arduino.pinMode(greenLedPin, Arduino.OUTPUT); // imposta il LED verde come output PIN  
    arduino.digitalWrite(greenLedPin, Arduino.LOW); // spegne il LED verde  
    arduino.pinMode(redLedPin, Arduino.OUTPUT); // imposta il LED rosso come output PIN  
    arduino.digitalWrite(redLedPin, Arduino.LOW); // spegne il LED rosso  
    arduino.pinMode(pirPin, Arduino.INPUT); // imposta il sensore PIR come input PIN  
    arduinoCheck=true;  
  }  
  catch(Exception e) {  
    println("Error: "+e.toString());  
    arduinoCheck=false;  
  }  
  
  delay(5000); // aspettiamo 5 secondi in modo che tutto venga inizializzato correttamente  
}
```

Stampa in console qualcosa come:
COM3 COM4 COM5 COM6 COM8

Se Arduino corrisponde a COM5
nell'elenco precedente

Uno sguardo al codice (III)

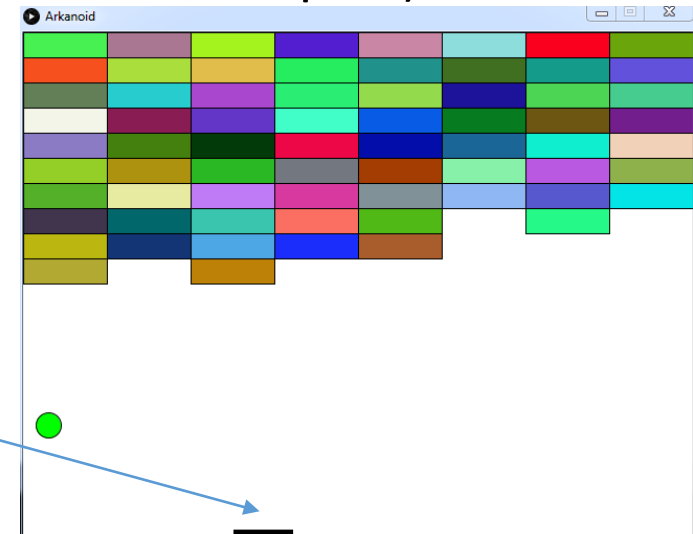
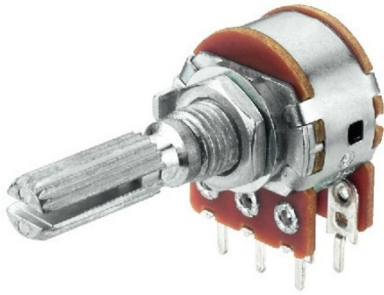
```
void draw() {  
  ...  
  if(arduinoCheck) {  
    int pirSignal=arduino.digitalRead(pirPin);  
  
    if(pirSignal==Arduino.HIGH) {  
      arduino.digitalWrite(redLedPin, Arduino.HIGH);  
      arduino.digitalWrite(greenLedPin, Arduino.LOW);  
      ...  
    }  
    else {  
      arduino.digitalWrite(greenLedPin, Arduino.HIGH);  
      arduino.digitalWrite(redLedPin, Arduino.LOW);  
      ...  
    }  
  }  
  ...  
}
```

Se rilevo un movimento,
accendo il led rosso
e spengo il led verde...

...altrimenti, faccio il viceversa.

Progetto 2: usare un potenziometro come "gamepad" per Arkanoid

- Un potenziometro è un partitore di tensione regolabile.
- Può essere utilizzato come un reostato, ovvero, permette di ottenere una resistenza variabile.
- L'idea è di tradurre il valore in uscita (ottenuto ruotando la manopola) in uno spostamento per la paletta del gioco Arkanoid.



- Useremo anche tre LED per rappresentare le vite rimanenti al giocatore (LED spento=vita disponibile, LED acceso=vita persa).
- **La tastiera serve solo per: F1 → Nuova partita, F2 → Pausa/Riprendi**

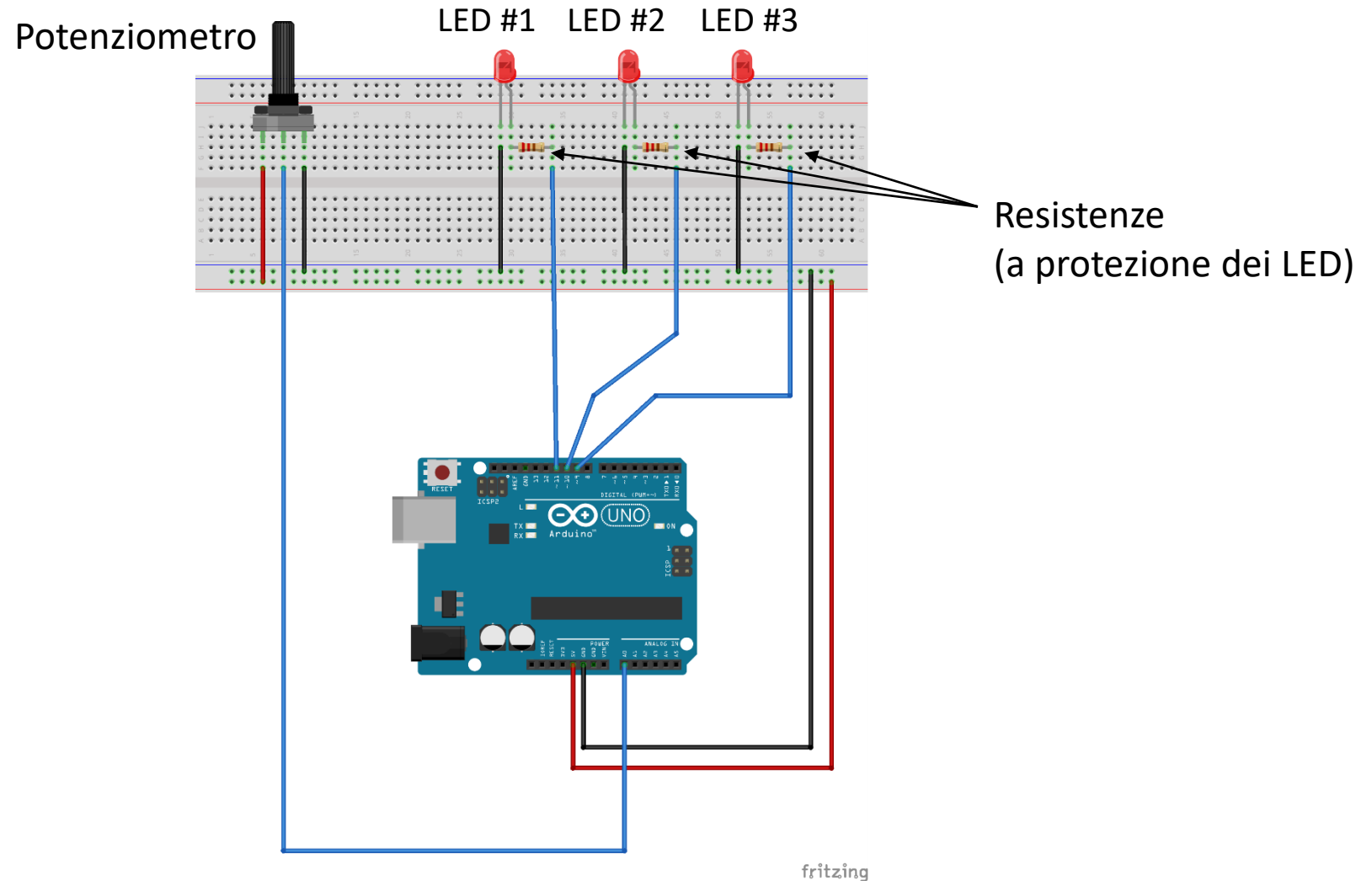
Schema del progetto

Colore cavi:

Rosso: tensione (+5V)

Nero: terra (ground)

Blu: segnale



Uno sguardo al codice (I)

```
import processing.serial.*;
import cc.arduino.*;

...

int potpin = 0; // potenziometro
int firstled = 9, secondled = 10, thirdled = 11; //
i tre led che rappresentano le vite del giocatore
Arduino arduino; // oggetto di tipo Arduino: serve a
dialogare con la board

boolean arduinoCheck = false; // booleano: true ->
Arduino collegato, false -> Arduino assente
```

Uno sguardo al codice (II)

```
void setup() {  
    ...  
    try {  
        println((Object[])Arduino.list());  
        arduino = new Arduino(this, Arduino.list()[0], 57600); // inizializza l'oggetto arduino  
        arduino.pinMode(firstled, Arduino.OUTPUT);           // imposta il primo LED come output PIN  
        arduino.digitalWrite(firstled, Arduino.LOW);          // spegne il primo LED  
        arduino.pinMode(secondled, Arduino.OUTPUT);           // imposta il secondo LED come output PIN  
        arduino.digitalWrite(secondled, Arduino.LOW);          // spegne il secondo LED  
        arduino.pinMode(thirdled, Arduino.OUTPUT);            // imposta il terzo LED come output PIN  
        arduino.digitalWrite(thirdled, Arduino.LOW);           // spegne il terzo LED  
        arduinoCheck = true;  
    }  
    catch(Exception e) {  
        println("Error: "+e.toString());  
        arduinoCheck = false;  
    }  
    ...  
}
```

Stampa in console qualcosa come:
COM3 COM4 COM5 COM6 COM8

Sostituire con l'indice
corrispondente al numero di
porta corretto sul proprio PC
(ad esempio 0 per COM3).

Uno sguardo al codice (III): righe 255—274 (funzione moveBall())

```
...
if(bally+ballRadius>=WINDOW_ROWS) { // rimbalzo sul fondo -> vita persa (accendo un LED diverso
per ognuna delle tre vite perse)
    life--;
    print("Vite rimanenti: "+life+"\n");
    if(arduinoCheck) {
        switch(life) {
            case 0: // 0 vite, game over -> tutti i LED accesi
                // Esercizio: scrivere il codice per accendere i LED
                break;
            case 1: // 1 vita rimasta -> primi due LED accesi, terzo LED spento
                // Esercizio: scrivere il codice per accendere/spegnere i LED
                break;
            case 2: // 2 vite rimaste -> primo LED acceso, secondo e terzo LED spenti
                // Esercizio: scrivere il codice per accendere/spegnere i LED
                break;
        }
    }
}
...
```

Uno sguardo al codice (IV): righe 291—299 (funzione draw())

```
if(arduinoCheck) {  
    // lettura del valore del potenziometro  
    int val=...; // Completare il codice  
    // mappatura del valore in termini di ascisse della finestra  
    int mappedVal=(int)map(val,0,1023,...); // Completare  
    if(DEBUG) println("*****"+mappedVal+"*****");  
  
    if(matchStatus==1) { // se la partita è in corso  
        paddleX=...; // Completare: spostare la paletta  
    }  
}
```

Uno sguardo al codice (V): righe 323—337 (funzione keyPressed())

```
...
if(keyCode==KeyEvent.VK_F1) { // nuova partita: l'utente ha premuto F1
    ...
    life=3;
    points=0;
    ...
    if(arduinoCheck) { // spengo i 3 LED: le vite vengono ripristinate.
        // Completare il codice
    }
}
...
}
```