

# Fisica con Arduino

Introduzione ad Arduino e alla sua programmazione

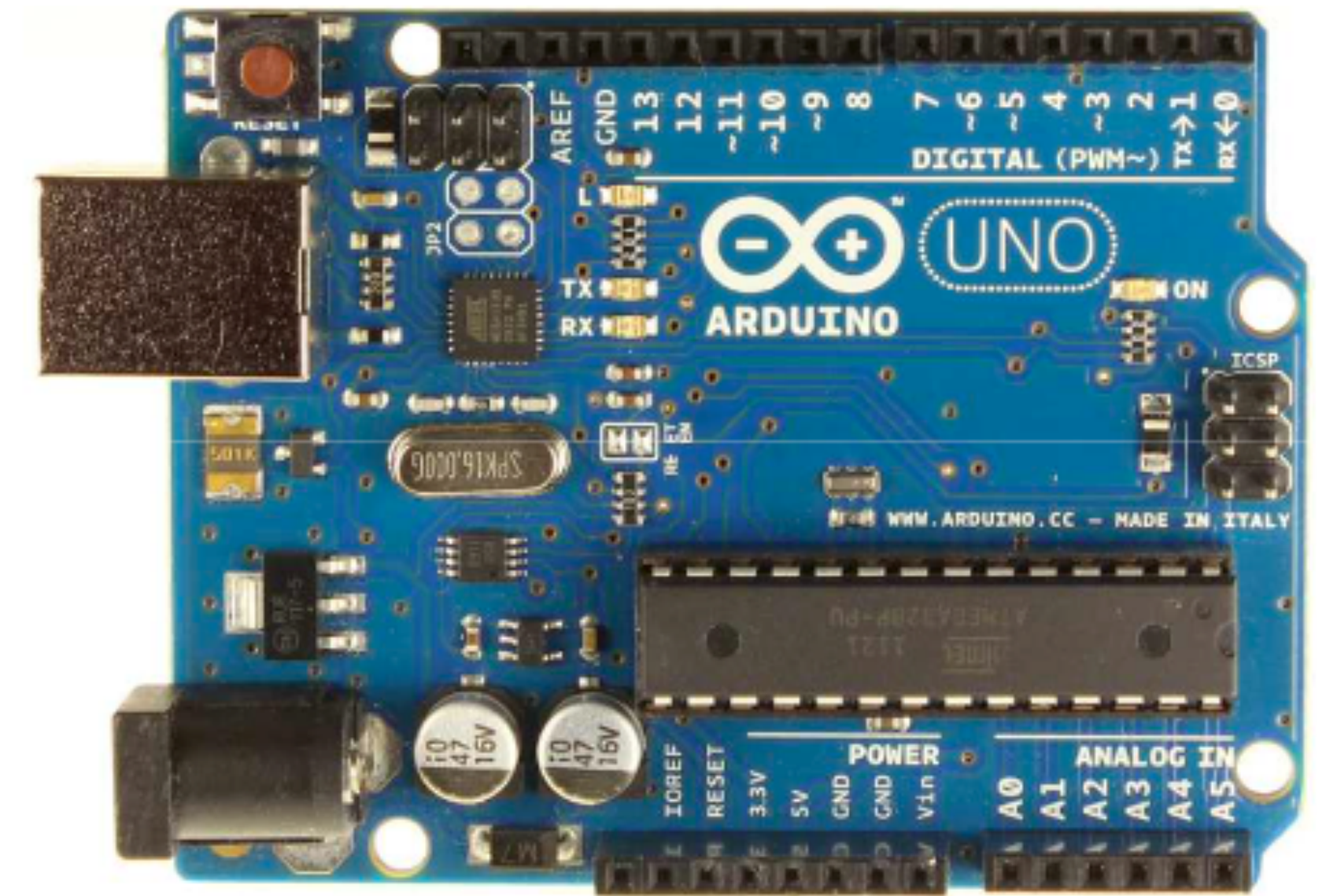
**Physics Hackathon**

02 Settembre 2024

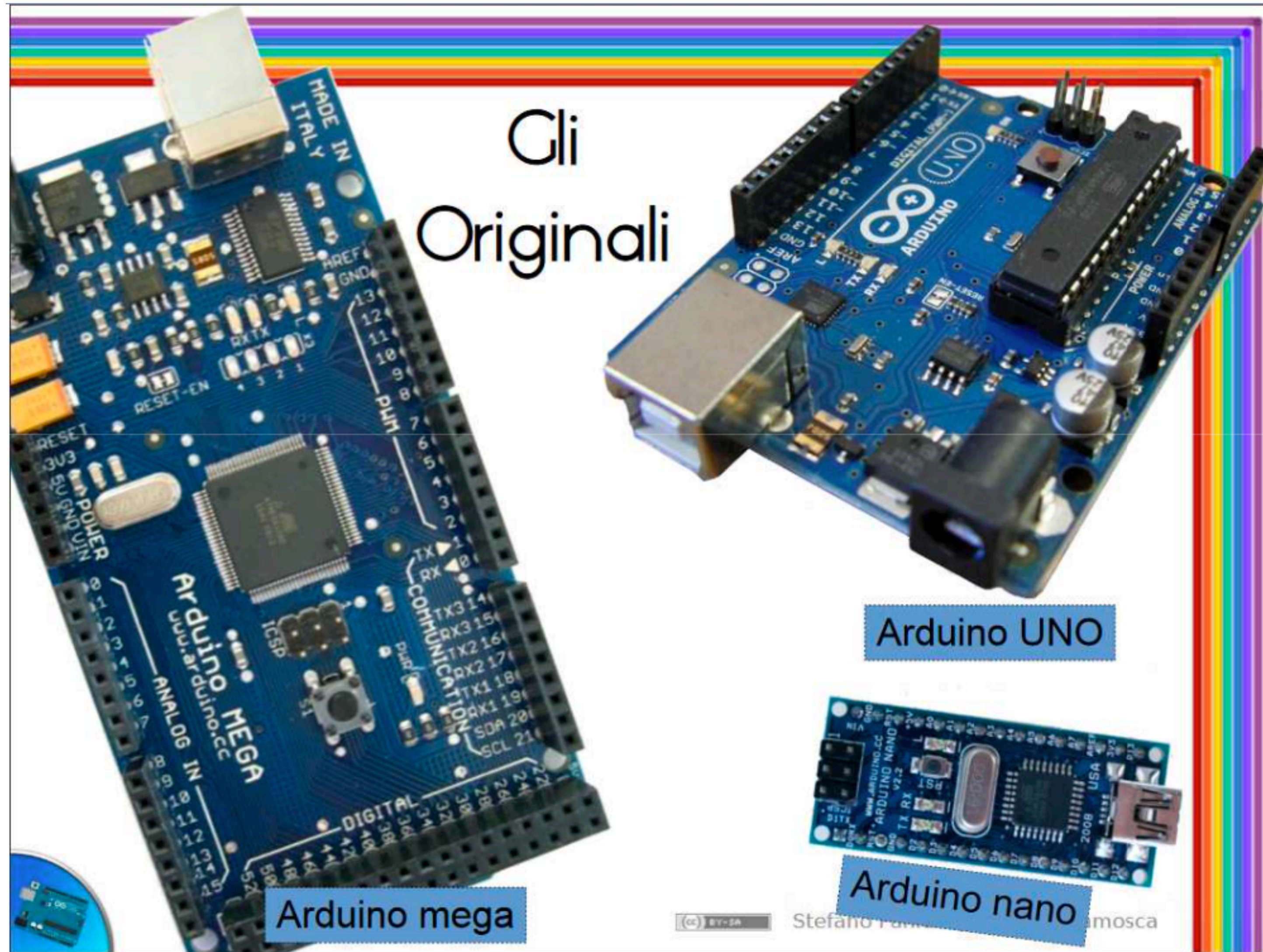
# Introduzione

# Cos'è Arduino

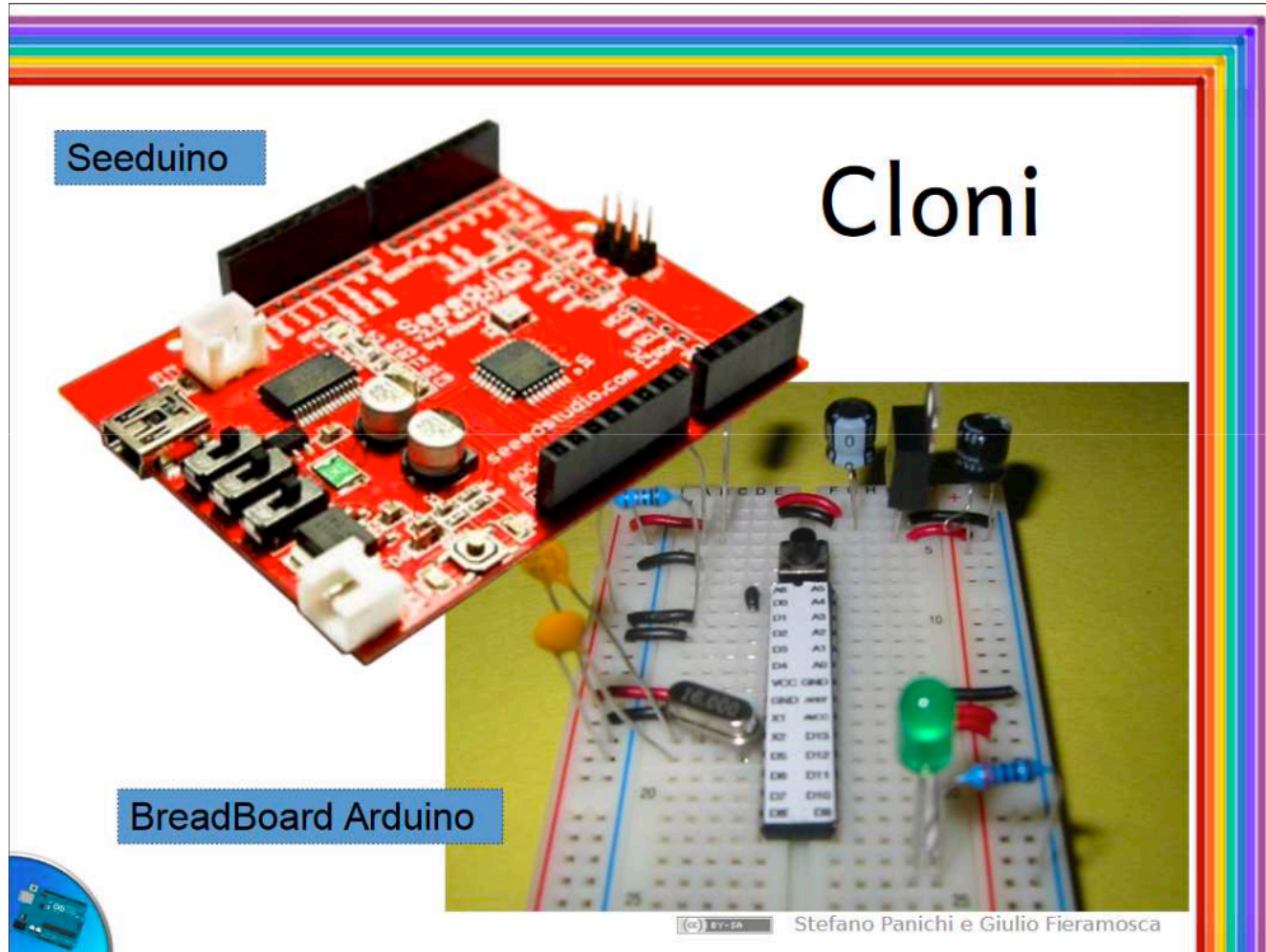
- Arduino è una **scheda elettronica di piccole dimensioni** (comandata da un microprocessore) che presenta una serie di ingressi e uscite (pin) analogici o digitali che permettono di inviare e ricevere segnali.
- La scheda nasce nel 2005 grazie al lavoro fatto di Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis a Ivrea. Esistono diverse schede Arduino ma quella da noi usata è l'Arduino UNO REV 3 il cui processore è un ATmega328P. Tale scheda ha una memoria di 32Kb.



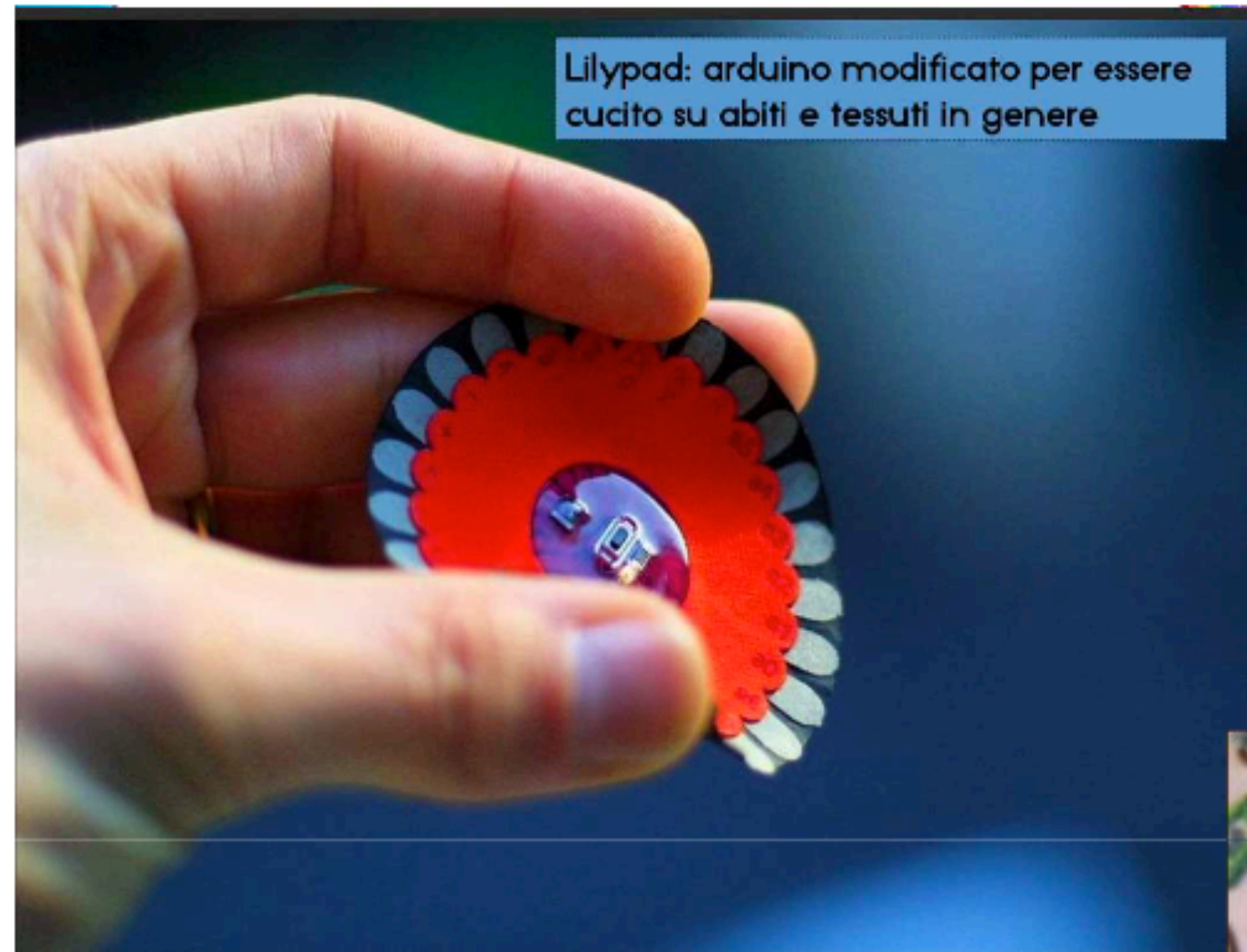
# Diversi tipi di Arduino



# Diversi tipi di Arduino



# Diversi tipi di Arduino



# Diversi tipi di Arduino



# Cos'è un micro-controllore

- Si tratta di un componente elettronico **programmabile** che consente di sviluppare dispositivi "intelligenti" a basso costo
- È un piccolo computer con velocità e memoria ridotte, quindi consumi inferiori





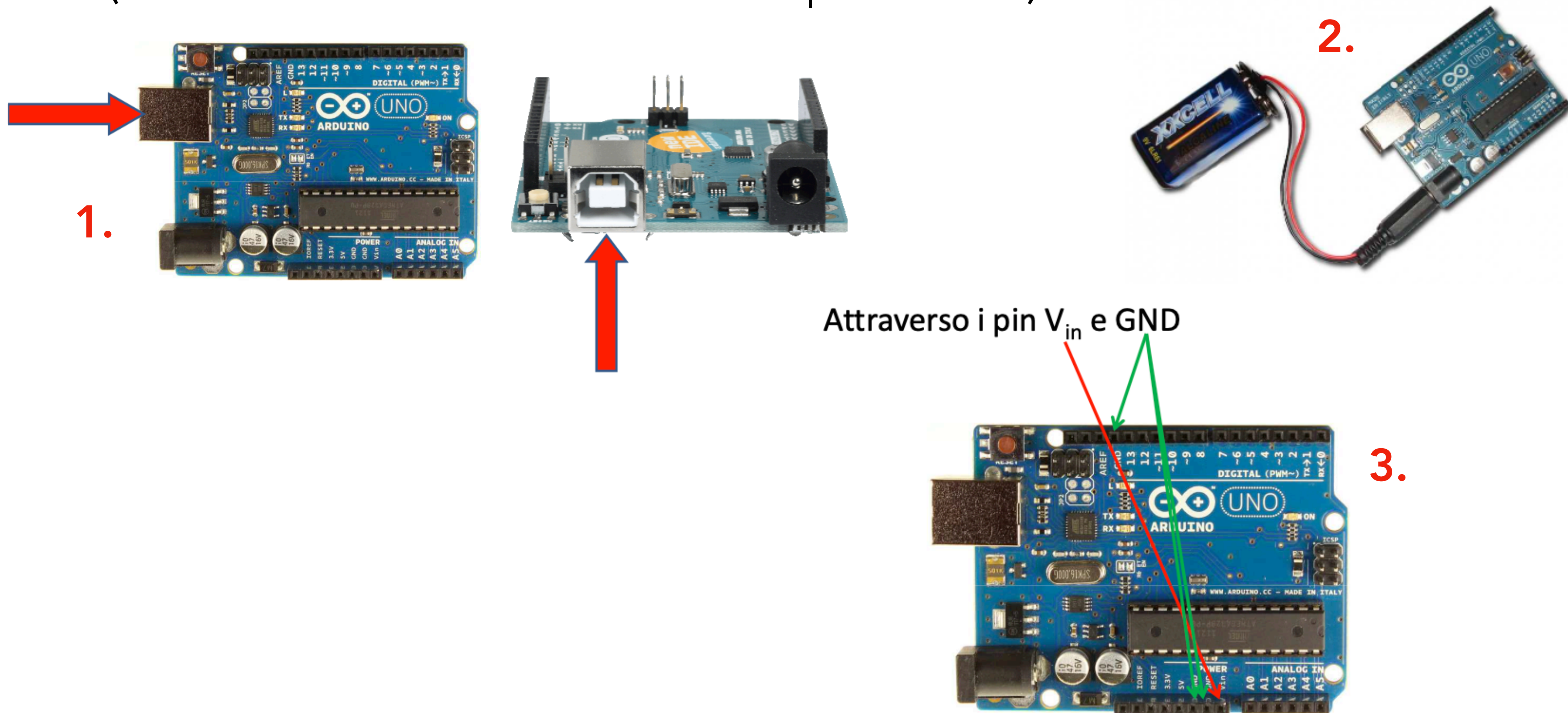
# Applicazioni

- Le più disparate. In internet esistono centinaia di progetti
- Nel nostro caso useremo Arduino per leggere sensori e condurre esperimenti di fisica
- Dove si può acquistare:
  - SITO UFFICIALE: <https://www.arduino.cc>
  - In internet, ogni sito di elettronica lo vende. Gli originali costano 20 – 30 euro. Non originali (cinesi) intorno a 10 euro.
- Alcuni siti
  - <http://www.html.it/guide/guida-arduino/>
  - <http://www.mauroalfieri.it/corso-arduino-on-line.html>
  - <http://share.dschola.it/castigliano/elettronici/5f/Materiali/ARDUINO/Tiziana Marsella - Programmare Arduino.pdf>

# Alimentazione e pins

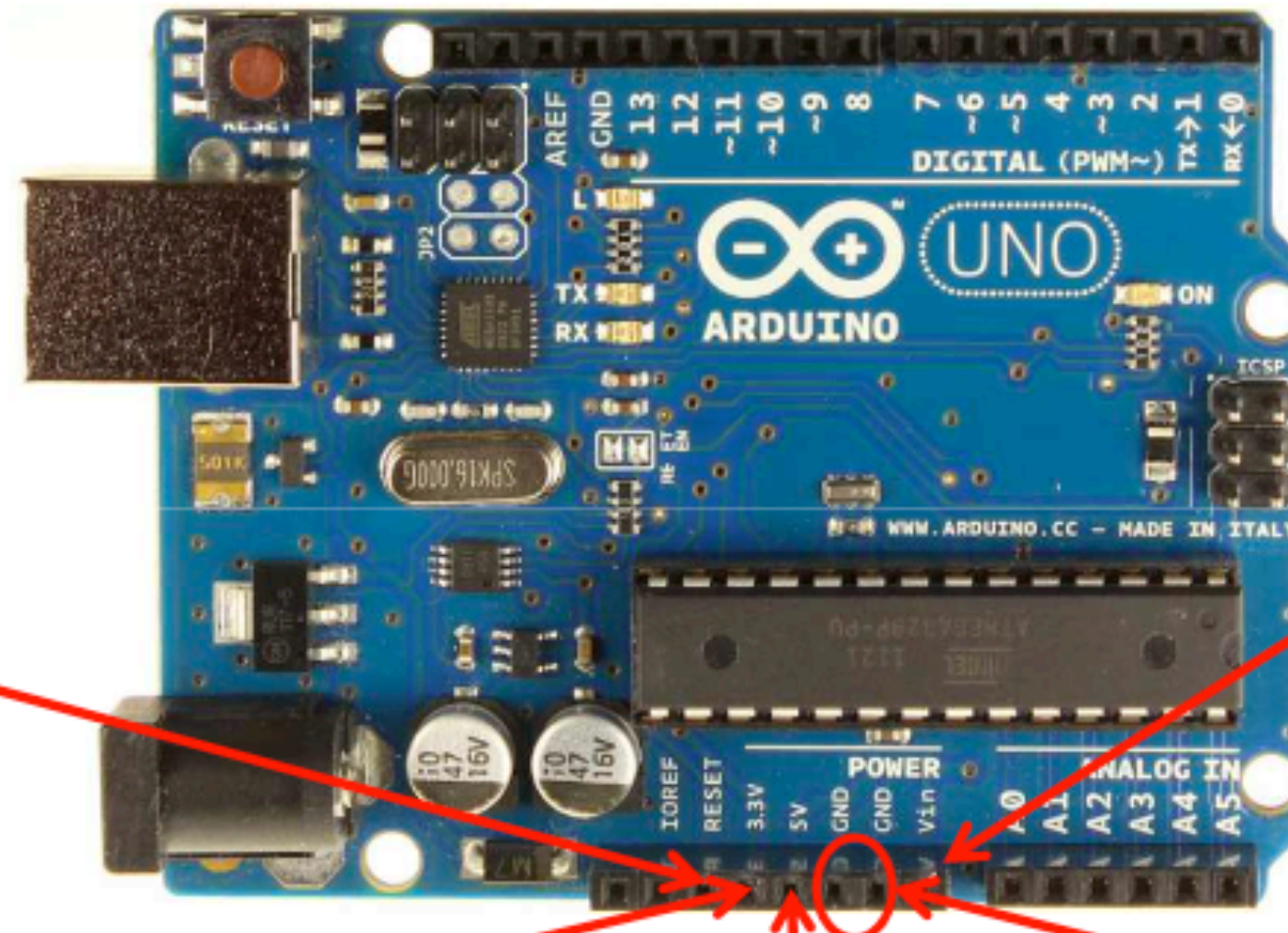
# Come alimentare Arduino

- La scheda è alimentata con una tensione di 5 V e una corrente di 500 mA (valori ideali di funzionamento del processore)



# Power pins

RESET: è un pin che se collegato a un pin digitale di Arduino può riavviare la scheda (la scheda viene riavviata quando il pin digitale, che si trova normalmente allo stato HIGH, viene portato al valore LOW)



pin Vin: si può usare per alimentare la scheda

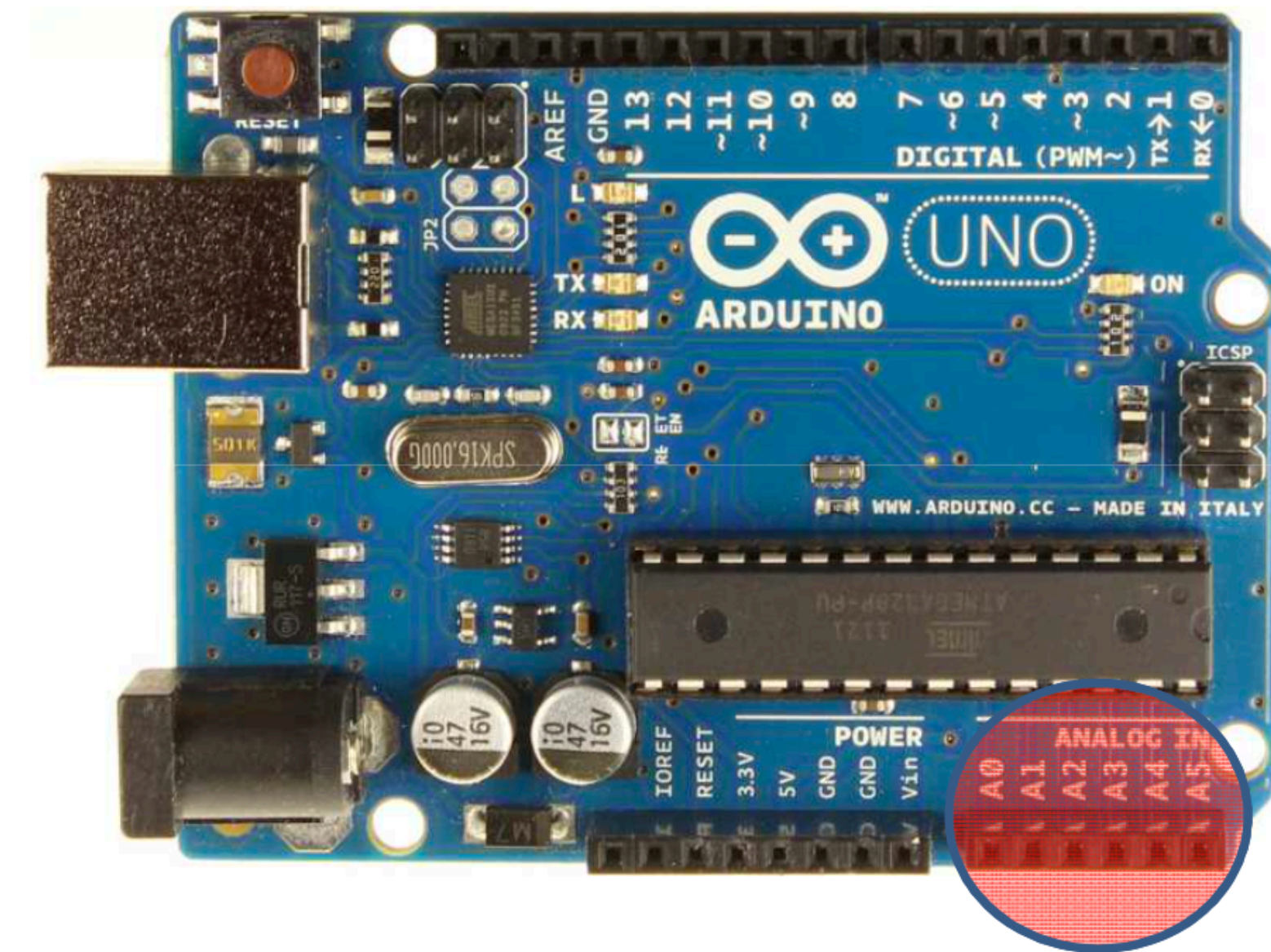
pin 3.3 V: è un'uscita che fornisce una tensione costante di 3.3 V e una corrente massima di 150 mA

pin 5 V: è un'uscita che fornisce un potenziale costante di 5 V (in teoria non ci sono limiti di corrente)

pin GND: sono due pin di massa della scheda

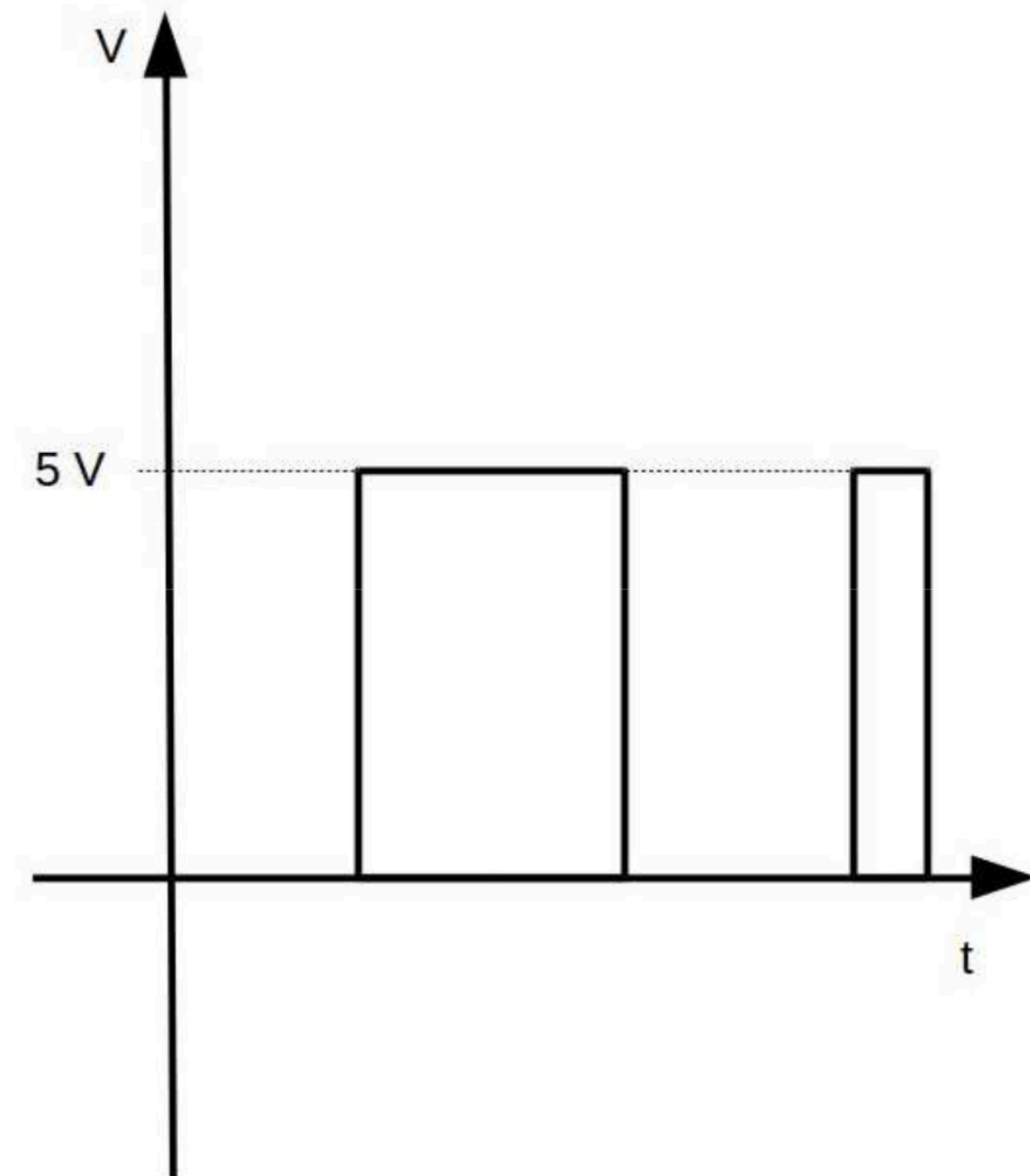
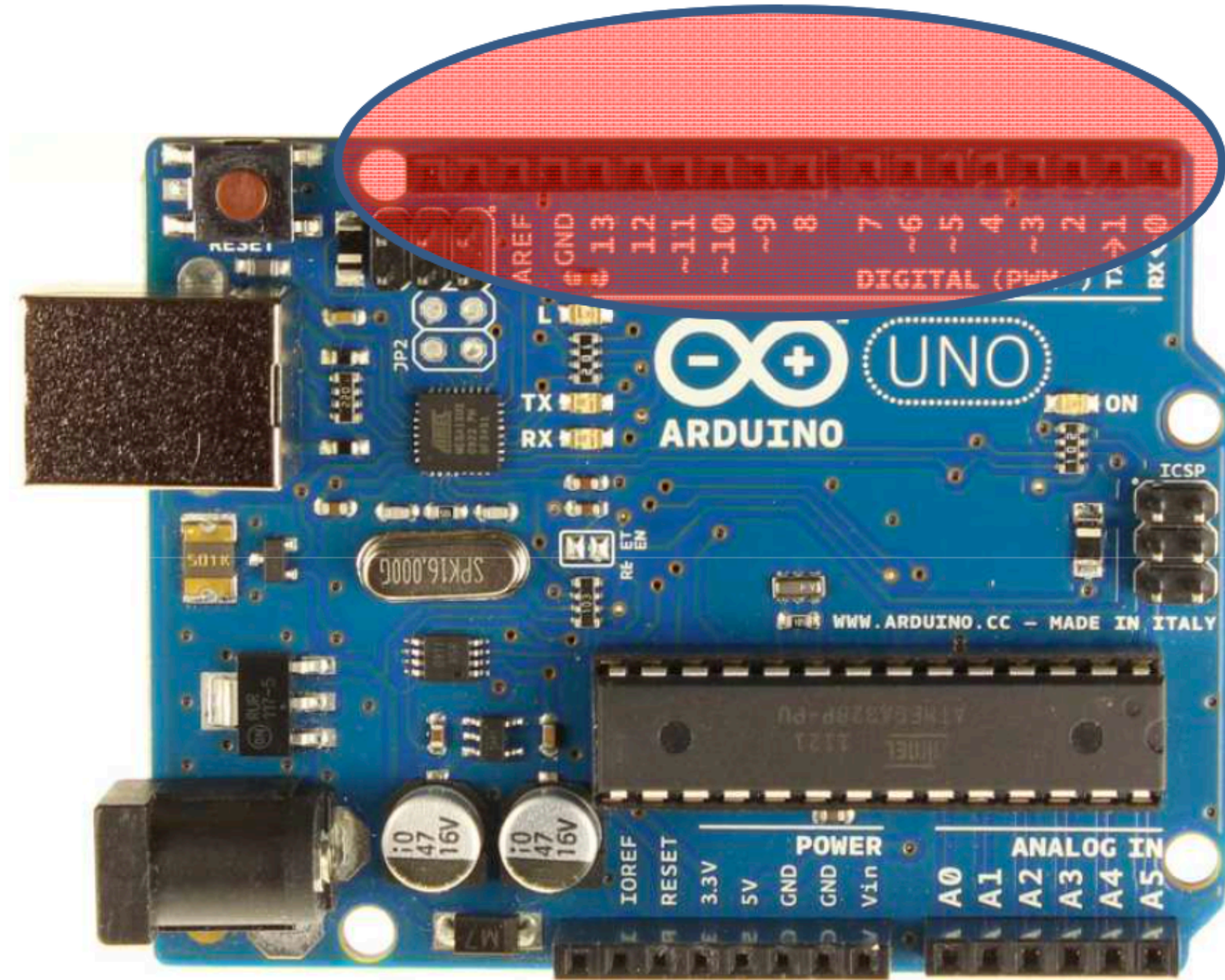
# Pin analogici

- Arduino UNO R3 presenta 6 pin analogici. **Tali pin sono di input.**
- Sono 6 convertitori analogico- digitale (ADC) con risoluzione di 10 bit. Ogni convertitore analogico digitale ha quindi un numero di canali pari a:  
 $2^{10} = 1024$
- Gli ADC ci permettono di **leggere una tensione nel range 0 - 5 V.**
- Il comando di Arduino analogRead, che permette di fare questa operazione, ci dà il risultato della misura in canali
  - un valore compreso fra 0 e 1023
- Per convertire il valore misurato in volt possiamo fare:



$$\text{tensione(V)} = \text{tensione(Ch)} \times \frac{5}{1023}$$

# Pin digitali

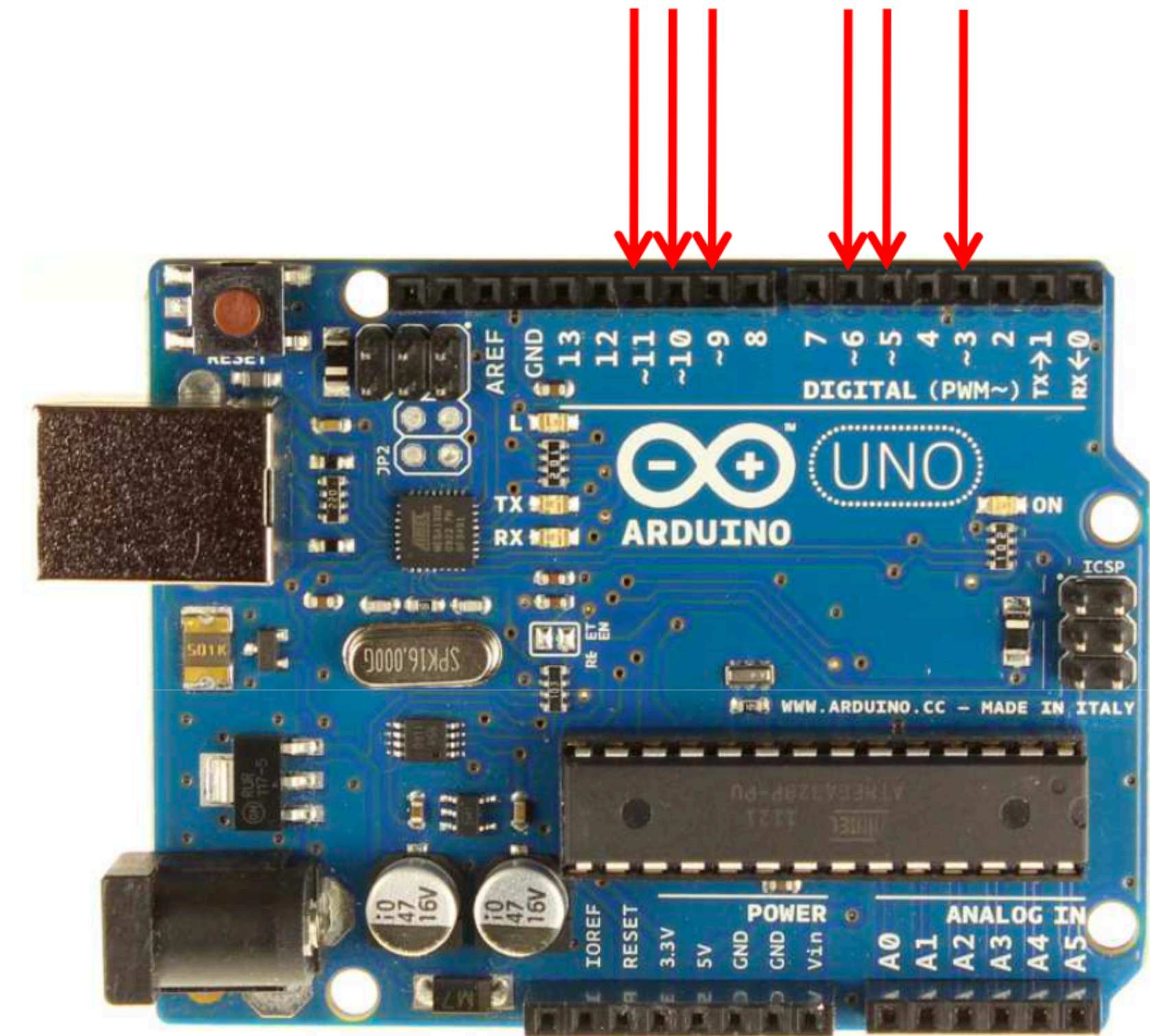


# Pin digitali

- I pin digitali sulla scheda Arduino UNO REV3 sono 13. **Tali pin possono essere utilizzati sia in input che in output**
  - va specificato attraverso il comando *pinMode*
- Quando sono utilizzati come output essi possono essere immaginati come degli "interruttori" e possono avere due stati:
  - **LOW**: il pin digitale è "spento" e la tensione in uscita è 0 V
  - **HIGH**: il pin digitale è "acceso" e la tensione in uscita è 5 V
- La corrente massima in uscita è di 40 mA
- Lo stato dei pin digitali viene cambiato con il comando ***digitalWrite***

# Pin digitali - Pulse Width Modulation (PWM)

- I pin digitali PWM sono quelli contrassegnati dal simbolo tilde [~]
- Nell'Arduino UNO REV3 sono i pin: 3, 5, 6, 9, 10, 11)
- I pin PWM sono dei pin digitali che possono fornire in output una tensione variabile da 0 V a 5 V
- **Anche in questo caso la tensione va scritta in canali e non in volt. Per farlo si utilizza il comando analogWrite.**
- In questo caso i canali sono 256 quindi la relazione che lega la tensione in canali a quella in volt è la seguente:

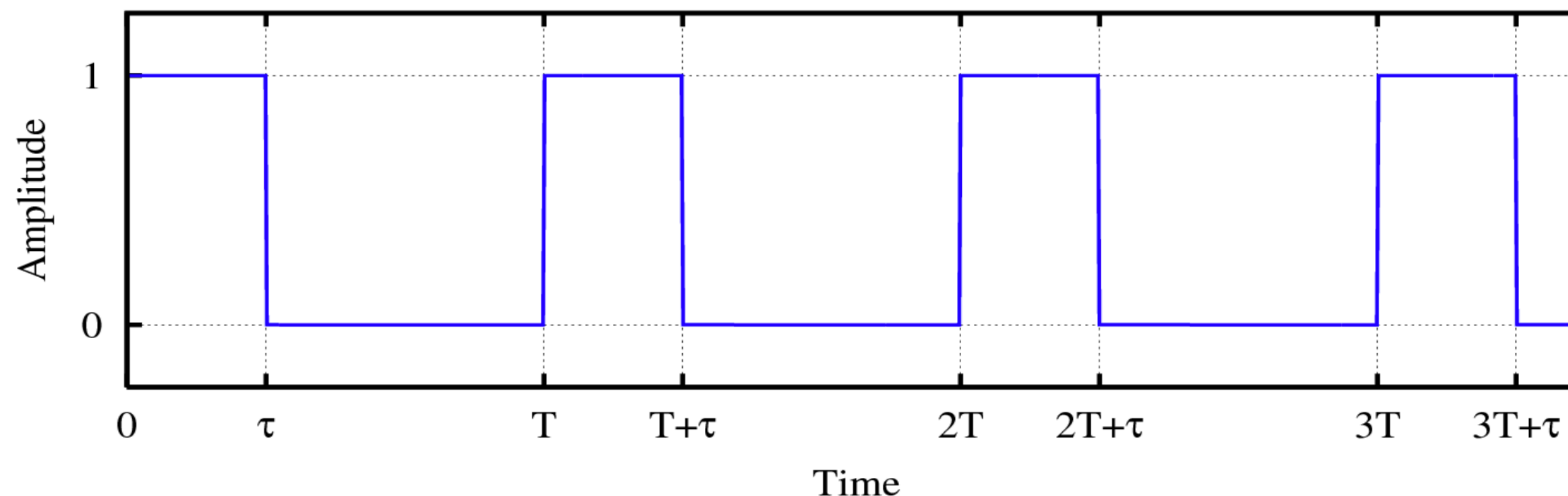


$$\text{tensione(Ch)} = \text{tensione(V)} \times \frac{255}{5}$$

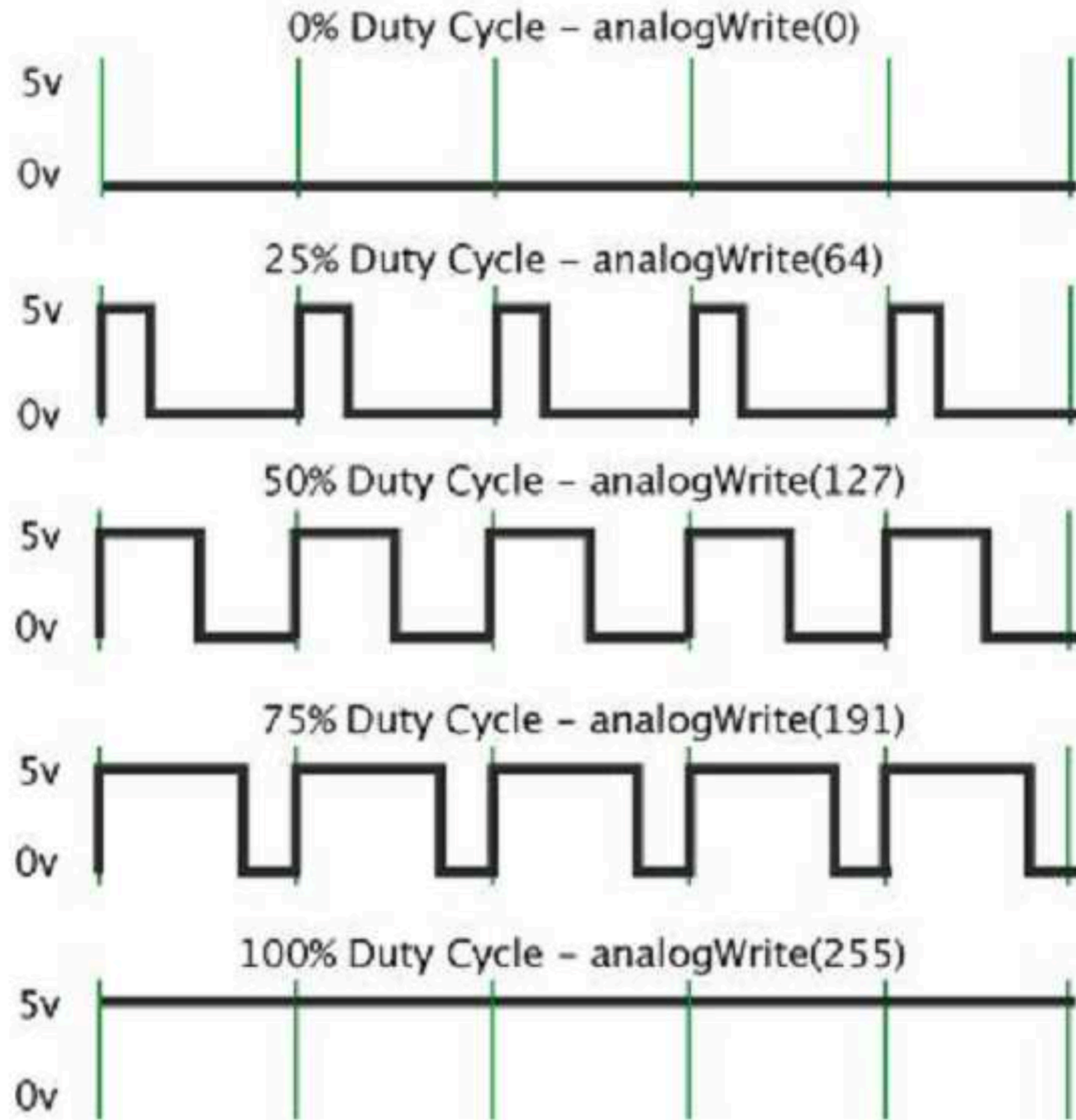


# Pin digitali - Pulse Width Modulation (PWM)

- La tensione viene modulata attraverso il metodo della **Pulse Width Modulation** (modulazione di larghezza di impulso)
- Esso consiste in una serie di impulsi ad una determinata frequenza (e quindi ad un determinato periodo  $T$ ) che restano accesi per un tempo  $\tau \leq T$
- In base al valore di  $\tau$  abbiamo quindi un diverso valore della tensione in uscita



# Pulse Width Modulation



# Come programmare Arduino

# L'ambiente di sviluppo IDE

- Per sviluppare il software per far funzionare la scheda Arduino, si utilizza l'ambiente di sviluppo (IDE) che ci permette di scrivere, compilare e trasferire i nostri programmi sulla scheda.
- <https://www.arduino.cc/en/Main/Software>

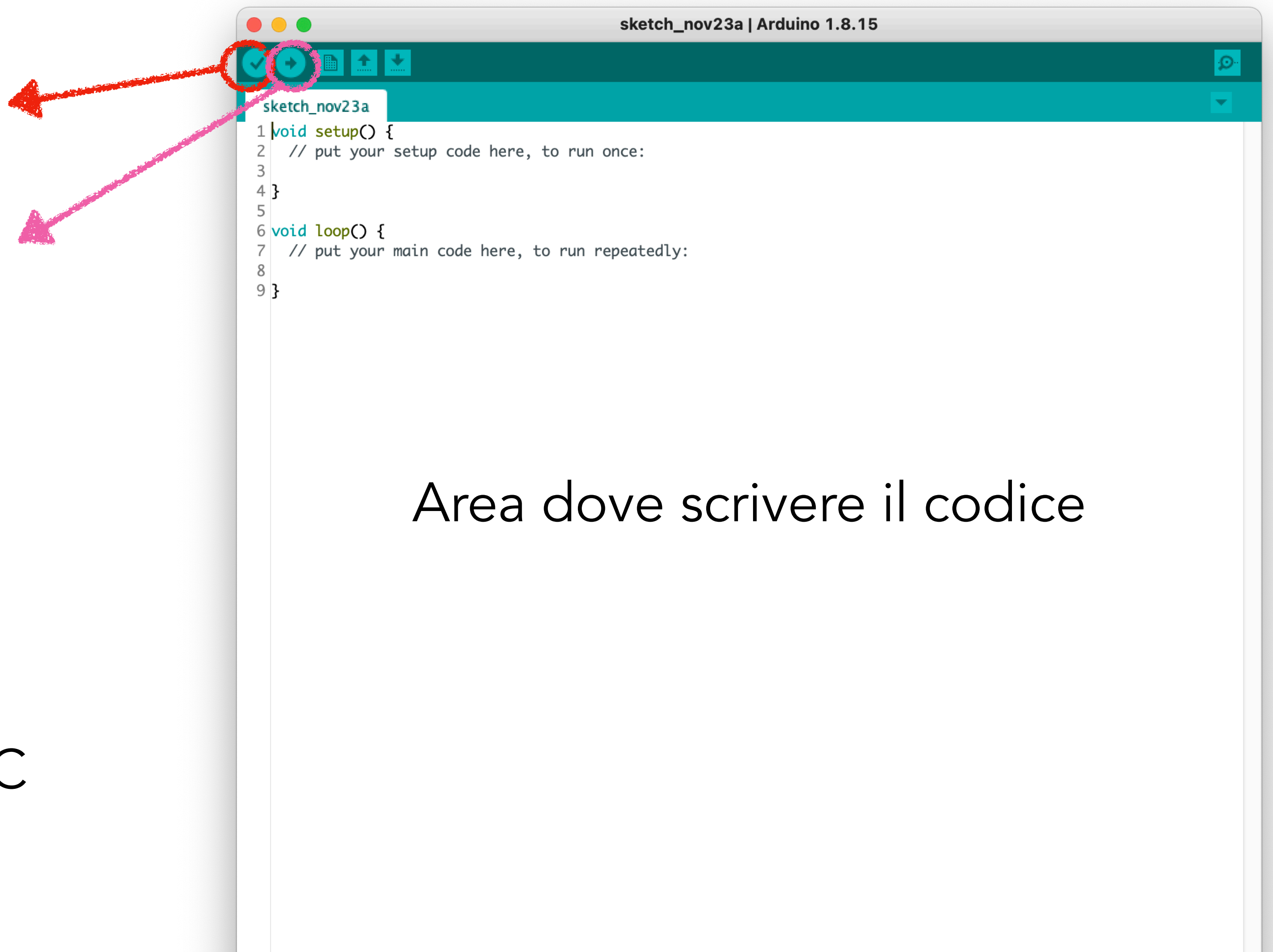
```
sketch_nov23a | Arduino 1.8.15
sketch_nov23a
1 void setup() {
2 // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7 // put your main code here, to run repeatedly:
8
9 }
```

1 Arduino Uno on /dev/cu.usbmodem1411301

**Compila** il codice, verifica l'esattezza della sintassi

**Carica** il codice compilato sulla scheda tramite USB

- La scheda Arduino viene programmata in un linguaggio molto simile al C utilizzando l'Arduino IDE



Area dove scrivere il codice

Barra degli errori

# Se il caricamento fallisce?

**RICORDA:** una volta aperto il programma è opportuno selezionare subito il tipo di scheda utilizzata dal menù *strumenti* → *scheda* → *Arduino uno* e la porta seriale dal menù *strumenti* → *porta*.



**ATTENZIONE:** a volte nel menù delle porte seriali sono presenti più porte quindi l'unico modo che si ha per vedere qual è quella giusta è selezionarne una e provare a caricare il programma.

Se il caricamento va a buon fine la porta scelta è quella giusta.  
Se il caricamento non va a buon fine si prosegue a tentativi!!!

# Struttura di uno *sketch* per Arduino



```
Blink | Arduino 1:1.0.5+dfsg2-2
File Modifica Sketch Strumenti Aiuto
Blink $
#define led 13          Definizioni e dichiarazione variabili
// int led = 13;

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

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

Compilazione terminata.
```

Uno sketch per Arduino si divide in tre parti:

## ***Definizione e dichiarazione variabili***

In questa parte del programma si possono assegnare dei nomi ai pin analogici o digitali usati e si possono inoltre dichiarare le variabili (in realtà le variabili possono essere dichiarate in qualsiasi parte del programma purché sia prima del loro utilizzo).

# Struttura di uno *sketch* per Arduino

```
Blink | Arduino 1:1.0.5+dfsg2-2
File Modifica Sketch Strumenti Aiuto
Blink
Definizioni e dichiarazione variabili
#define led 13
// int led = 13;

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

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

Compilazione terminata.
```

Uno sketch per Arduino si divide in tre parti:

## ***void setup***

È un ciclo che Arduino esegue una sola volta e in cui si definisce se i pin digitali usati sono di input o di output e il loro stato iniziale (LOW o HIGH).

In questo ciclo di definiscono inoltre gli ADC utilizzati e si inizializza anche la comunicazione con la porta seriale.



# Struttura di uno *sketch* per Arduino



```
Blink | Arduino 1:1.0.5+dfsg2-2
File Modifica Sketch Strumenti Aiuto
Blink $
#define led 13          Definizioni e dichiarazione variabili
// int led = 13;

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

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

Compilazione terminata.
```

Uno sketch per Arduino si divide in tre parti:

## ***void loop***

È un ciclo che viene eseguito dalla scheda infinite volte e che contiene tutte le istruzioni che la scheda deve eseguire.

# Note generali sulla programmazione

- Tutti i comandi devono terminare con ;
- Le parentesi graffe delimitano un blocco di comandi {}
  - Ad ogni parentesi aperta deve sempre corrispondere una parentesi chiusa
- Nella stesura di programmi è buona norma inserire commenti
  - Tutto quello compreso tra i simboli /\* e \*/ viene considerato come commento
  - Tutto quello che va dal simbolo // fino alla fine della riga è un commento.
- Per i comandi vedi Arduino – reference (in inglese) o Elementi base del linguaggio di programmazione di Arduino

# Definizione e dichiarazione di variabili

- Definizioni: si ha la possibilità di assegnare un nome ai pin di Arduino utilizzati utilizzando il seguente comando: ***#define nomepin numeropin***
- Il nomepin è arbitrario e può contenere sia lettere che numeri mentre il numeropin è quello riportato sulla relativa porta di Arduino
  - Ad esempio si può avere: ***#define chargePin 3***
- Nelle definizioni non è necessario distinguere tra porte analogiche e digitali in quanto tale distinzione sarà automatica nel momento in cui i nomi dati alle porte verranno inseriti nei vari comandi

# Definizione e dichiarazione di variabili

- Dichiarazione delle variabili: le variabili usate nel codice possono essere di diverso tipo e vanno dichiarate, per esempio:
  - **int** → intero a 16 bit, va da  $-32768$  a  $+32767$ ;
  - **unsigned int** → intero a 16 bit, va da  $0$  a  $+65535$ ;
  - **long** → intero a 32 bit, va da  $-2\,147\,483\,648$  a  $+2\,147\,483\,647$ ;
  - **unsigned long** → intero a 32 bit, va da  $0$  a  $+4\,294\,967\,295$ ;
  - **float** → numero in virgola mobile a 32 bit, va da  $-3.4028235 \cdot 10^{+38}$  e  $3.4028235 \cdot 10^{+38}$ ;
  - **double** → numero in virgola mobile a 64 bit (solo in alcune schede).
- E' buona norma inizializzare le variabili
  - **tipo\_di\_variabile nome\_variabile = valore\_iniziale ;**
- Come in C++ per i vettori la sintassi è per esempio:
  - **int vettore[100] ;**

# Cosa mettere nel *void setup*

- I comandi da mettere necessariamente nel ***void setup*** sono:
  - ***Serial.begin(9600)***
    - se si ha intenzione di stampare dei dati su file o sul monitor seriale. Tale comando inizializza la porta seriale
  - ***pinMode(nomeporta / numeroporta , INPUT / OUTPUT)***
    - se si usano delle porte digitali. Tale comando imposta le porte digitali usate o in INPUT o in OUTPUT.
  - ***bitClear(ADCSRA,ADPS0)***
    - se si usano delle porte analogiche per la lettura di una tensione. Tale comando esegue il reset dell'ADC e ne stabilisce la frequenza di campionamento
- Nel caso in cui si utilizzino delle porte digitali è opportuno specificare anche lo stato iniziale della porta (ovvero lo stato in cui si troverà la porta una volta usciti dal void setup); il comando è:
  - ***digitalWrite(nomeporta / numeroporta, HIGH / LOW)***
    - Tale comando serve a scrivere HIGH o LOW sulla porta digitale specificata

# Cosa mettere nel *void loop*

- Il void loop è il ciclo in cui diciamo ad Arduino **cosa fare**
- Le istruzioni che possiamo inserire sono tutte quelle del linguaggio C (**operazioni matematiche, cicli, ecc.**) insieme ad alcuni **comandi specifici di Arduino** che servono a leggere e scrivere su porte digitali o analogiche

# Comandi per porte digitali

- ***digitalWrite(nomeporta / numeroporta, HIGH / LOW)***
  - Comando per scrivere su una porta digitale
- ***digitalRead(nomeporta / numeroporta)***
  - Comando per leggere lo stato di una porta digitale
- ***pulseIn(nomeporta / numeroporta, HIGH/LOW)***
  - Tale comando ci da il tempo (in s) durante il quale una porta digitale è rimasta rispettivamente nello stato HIGH o LOW (es: pulseIn(4,HIGH) ci da in s il tempo che la porta digitale 4 resta nello stato HIGH)
  - Questa funzione è utile con il sensore a ultrasuoni che useremo negli esperimenti
- ***analogWrite(nomeporta / numeroporta, 0-255)***
  - Comando per scrivere su una porta digitale **PMW** Il valore 0 corrisponde a 0 V mentre il valore 255 corrisponde a 5 V

# Comandi per porte analogiche

- ***analogRead(nomeporta / numeroporta)***
  - Comando per leggere il valore di tensione in ingresso a una porta analogica
  - Il valore in uscita varia tra 0 (0 V) e 1023 (5V)



# Altre funzioni del *void loop*

- Funzioni che permettono di inserire un tempo d'attesa tra un'istruzione e la successiva
  - ***delay(variable/numero)***
    - aspetta un numero di millisecondi pari al valore della variabile o del numero messo fra parentesi tonde (deve essere un intero)
  - ***delayMicroseconds(variable/numero)***
    - aspetta un numero di microsecondi pari al valore della variabile o del numero messo fra parentesi tonde (deve essere un intero)
- Funzioni che permettono di contare il tempo trascorso da quando è stato lanciato lo sketch
  - ***millis()***
    - restituisce in millisecondi il tempo trascorso da quando è stato lanciato lo sketch
  - ***micros()***
    - restituisce in microsecondi il tempo trascorso da quando è stato lanciato lo sketch

# Programmazione Strutture principali (Refresher)

# Strutture principali

- I principali costrutti che utilizzeremo nei nostri sketch sono i seguenti:
  - **for**
    - è un ciclo che esegue tutte le istruzioni contenute al suo interno un numero fissato di volte.
  - **while**
    - è un ciclo che esegue tutte le istruzioni che sono al suo interno fino a quando è vera una determinata condizione logica.
  - **if**
    - è una struttura che esegue tutte le istruzioni contenute al suo interno una sola volta e solo se si verifica una data condizione logica.

## for loop

```
for(int i=0;i<100;i++)
{
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}
```

## while loop

```
int N=100;
int i=0;
while(i<N)
{
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
    i++;
}
```

## if statement

```
if( condizione logica)
{
    istruzioni
}
else if( condizione logica)
{
    istruzioni
}
else
{
    istruzioni
}
```

# Come fermare il *void loop*

Anche se non è davvero necessario

- Per fermare Arduino possiamo usare un “trucco” molto semplice... possiamo metterlo a non fare niente
- Possiamo quindi inserire (alla fine di tutte le istruzioni del void loop) un ciclo while da cui non esce mai e non mettere nessuna istruzione in questo ciclo
- `while(1<2) {}`

# Operatori

Di confronto

**==** UGUALE  
**!=** DIVERSO  
**>** MAGGIORE  
**>=** MAGGIORE UGUALE  
**<** MINORE  
**<=** MINORE UGUALE

Logici

**&&** AND  
**||** OR  
**!!** NOT

Come accedere ai dati

# Come fare in modo che Arduino inizi una misura quando vogliamo noi

- Una volta che il programma è stato caricato sulla scheda Arduino **inizia immediatamente l'esecuzione**
- Per far sì che lo sketch venga eseguito a partire da un determinato istante abbiamo almeno due modi:
  - Spegnere e riaccendere Arduino togliendo e rimettendo l'alimentazione alla scheda
  - Premere il tasto reset



# Come stampare i dati

- Supponiamo di avere 2 variabili x, y corrispondenti a due misure
- Vogliamo stampare i valori di x e y in modo da poterli accedere

```
void loop()
{
  x = analogRead(A1);
  y = analogRead(A2);

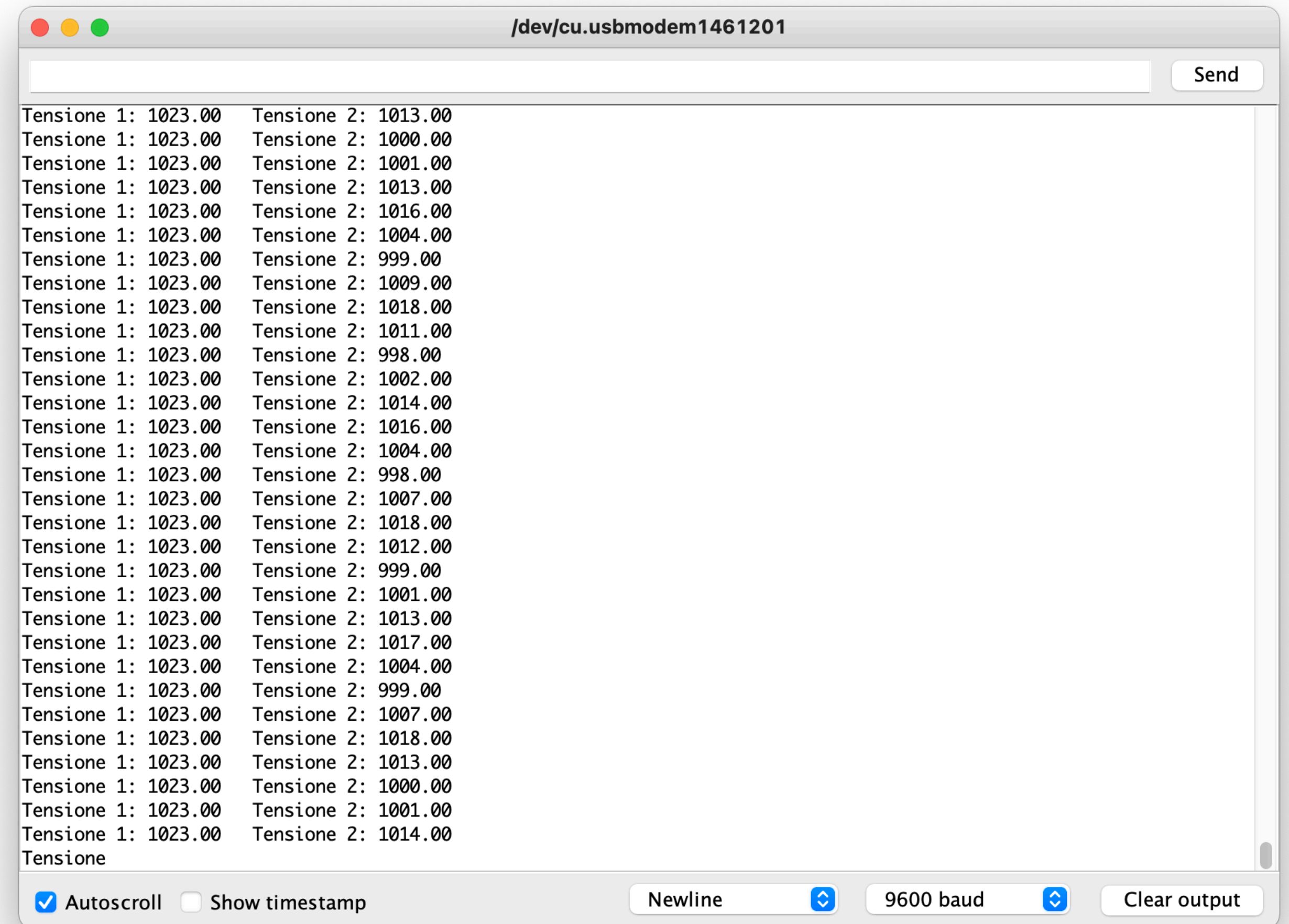
  Serial.print("Tensione 1: ");
  Serial.print(x);
  Serial.print(" ");
  Serial.print("Tensione 2: ");
  Serial.println(y);
}
```

Ci aspettiamo un output del tipo:

```
Tensione 1: 500 Tensione 2: 400
Tensione 1: 100 Tensione 2: 700
Tensione 1: 400 Tensione 2: 10
....
```

# Il monitor seriale

- Ci permette di accedere all'output della porta seriale e vedere ciò che viene stampato
  - i dati che compaiono sul monitor seriale non vengono memorizzati!
- Attenzione ad impostare il monitor seriale con la stessa velocità (baud rate) scelta nello sketch



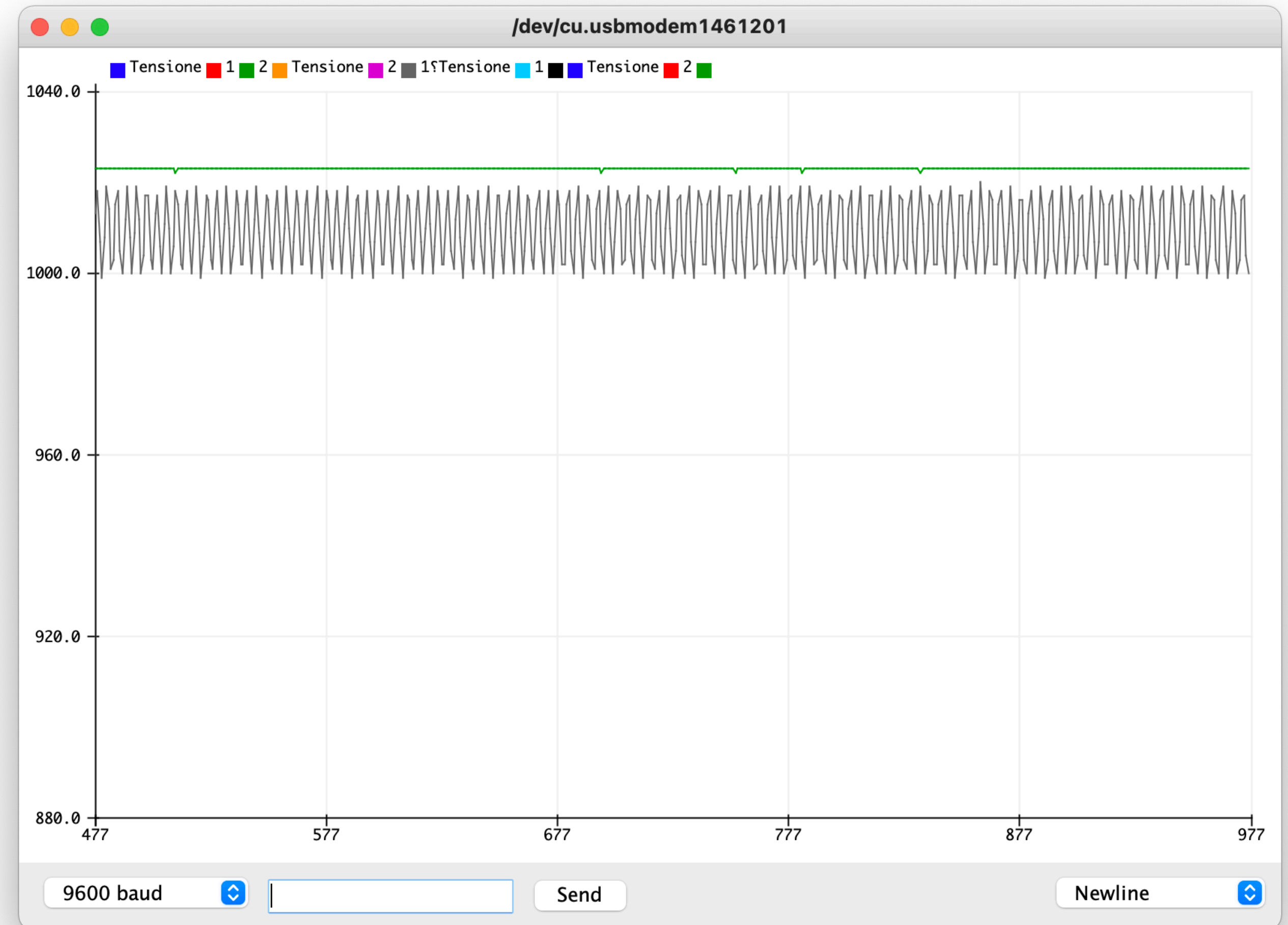
The screenshot shows a serial monitor window titled "/dev/cu.usbmodem1461201". The window contains a text area with the following output:

```
Tensione 1: 1023.00 Tensione 2: 1013.00
Tensione 1: 1023.00 Tensione 2: 1000.00
Tensione 1: 1023.00 Tensione 2: 1001.00
Tensione 1: 1023.00 Tensione 2: 1013.00
Tensione 1: 1023.00 Tensione 2: 1016.00
Tensione 1: 1023.00 Tensione 2: 1004.00
Tensione 1: 1023.00 Tensione 2: 999.00
Tensione 1: 1023.00 Tensione 2: 1009.00
Tensione 1: 1023.00 Tensione 2: 1018.00
Tensione 1: 1023.00 Tensione 2: 1011.00
Tensione 1: 1023.00 Tensione 2: 998.00
Tensione 1: 1023.00 Tensione 2: 1002.00
Tensione 1: 1023.00 Tensione 2: 1014.00
Tensione 1: 1023.00 Tensione 2: 1016.00
Tensione 1: 1023.00 Tensione 2: 1004.00
Tensione 1: 1023.00 Tensione 2: 998.00
Tensione 1: 1023.00 Tensione 2: 1007.00
Tensione 1: 1023.00 Tensione 2: 1018.00
Tensione 1: 1023.00 Tensione 2: 1012.00
Tensione 1: 1023.00 Tensione 2: 999.00
Tensione 1: 1023.00 Tensione 2: 1001.00
Tensione 1: 1023.00 Tensione 2: 1013.00
Tensione 1: 1023.00 Tensione 2: 1017.00
Tensione 1: 1023.00 Tensione 2: 1004.00
Tensione 1: 1023.00 Tensione 2: 999.00
Tensione 1: 1023.00 Tensione 2: 1007.00
Tensione 1: 1023.00 Tensione 2: 1018.00
Tensione 1: 1023.00 Tensione 2: 1013.00
Tensione 1: 1023.00 Tensione 2: 1000.00
Tensione 1: 1023.00 Tensione 2: 1001.00
Tensione 1: 1023.00 Tensione 2: 1014.00
Tensione
```

At the bottom of the window, there are several controls: a checked "Autoscroll" checkbox, an unchecked "Show timestamp" checkbox, a "Newline" dropdown menu, a "9600 baud" dropdown menu, and a "Clear output" button.

# Il plotter seriale

- Permette di plottare i valori stampati in funzione del tempo



# Come scaricare e salvare i dati

- Quando si apre il monitor seriale Arduino inizia di nuovo l'esecuzione dello sketch e stampa i dati sul monitor
- I dati si possono **copiare ed incollare in un foglio Excel**
  - Non ideale quando si devono selezionare grandi quantità di dati
  - Se i dati sono davvero molti non si ha la possibilità di vederli tutti sul monitor
- In alternativa si può usare uno **script Python per comunicare con la seriale** direttamente
  - Permette di salvare i dati in un file di testo

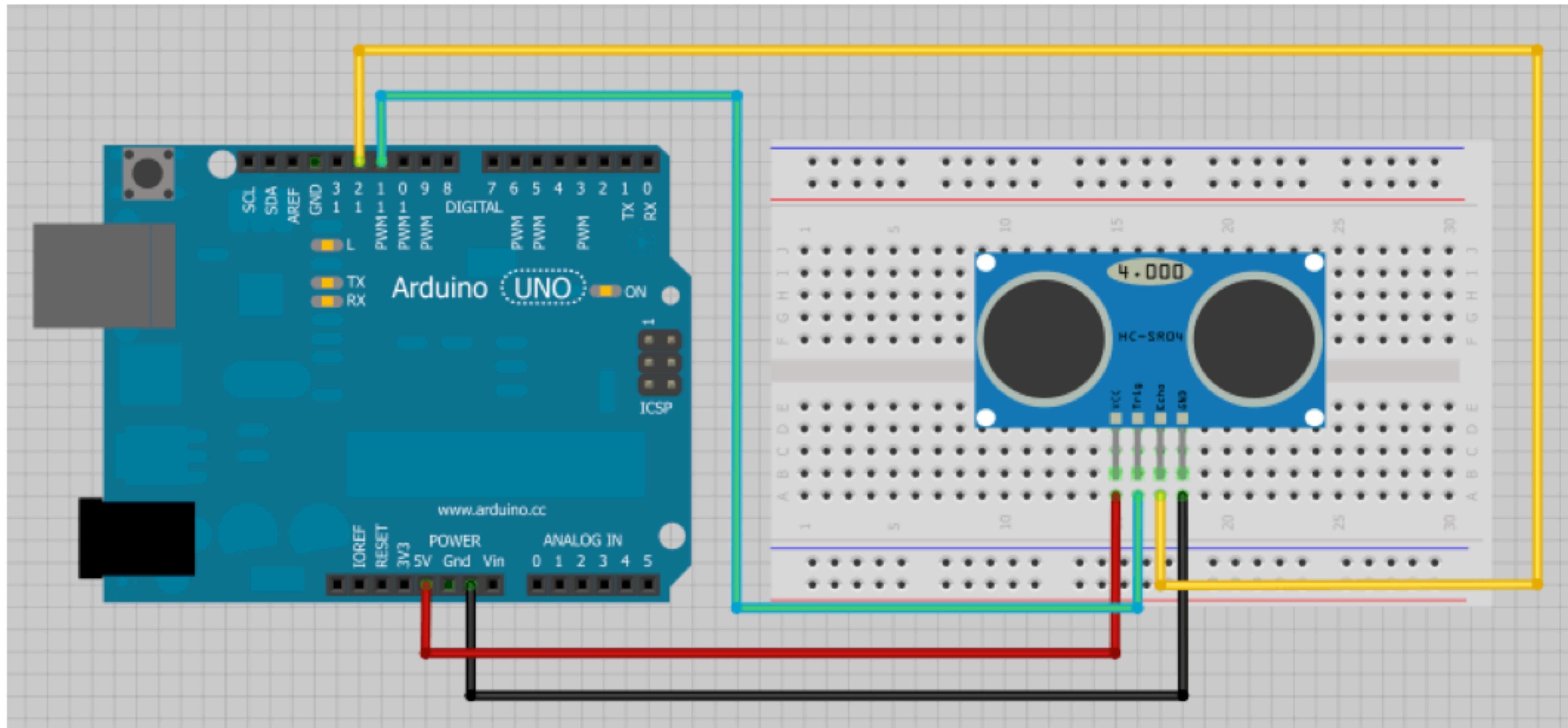
# I sensori

- I sensori che possono essere collegati alla scheda Arduino sono tantissimi e variano a seconda dello scopo dell'esperimento
  - **Sensori di posizione:** misurano la distanza tra il sensore e un ostacolo
  - **Sensori di prossimità:** restituiscono 0 o 1 a seconda che un corpo sia entro una certa distanza dal sensore
  - Sensori di movimento
  - Sensori di temperatura
  - Sensori di corrente
  - di pressione
  - di umidità
  - attuatori
  - ...

Un esempio:  
sensore di posizione ad ultrasuoni

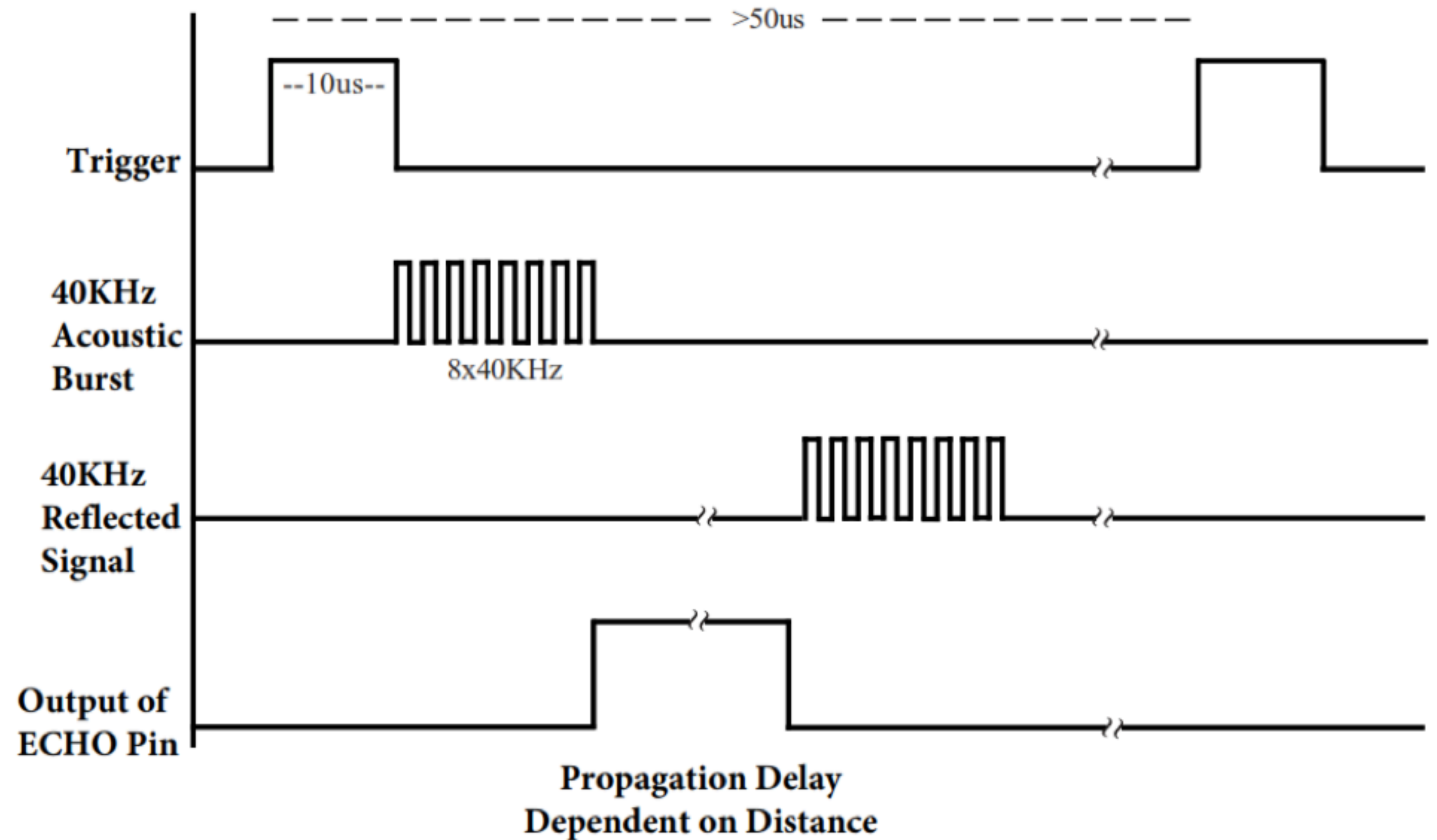
# Sensore ad ultrasuoni HC-SR04

- basato sulla riflessione del segnale a ultrasuoni, generato dal sensore stesso, sul primo ostacolo che tale segnale incontra lungo il suo cammino



- Si può misurare il tempo impiegato dal segnale per compiere il percorso di andata (sensore-ostacolo) e ritorno (ostacolo sensore)

- il sensore di posizione a ultrasuoni hr04 utilizza **due pin digitali, uno in input e uno in output**
- Si invia sulla porta digitale di trigger un impulso della durata  $>10\mu\text{s}$ 
  - tale impulso è lo start per l'invio del segnale
- Il sensore appena finito l'impulso invia un treno di 8 impulsi a ultrasuoni a una frequenza di 40 kHz e porta lo stato della porta di echo nel valore HIGH
- Nel momento in cui l'echo (il segnale riflesso dall'ostacolo) viene rilevato dal sensore la porta di echo viene portata di nuovo allo stato LOW





# Caratteristiche

- Costo: pochi €
- Max range: 4 m (massimo range effettivo: 2.5 m)
- Min range: 2 cm
- Errore:  $\pm 1$  cm
- Intervallo di tempo minimo tra una misura e l'altra: 25ms
- Intervallo di tempo minimo consigliato tra una misura e l'altra: 100 ms