

PROGETTO LAUREE SCIENTIFICHE FISICA

a.s. 2017-2018



Liceo Scientifico "G. Galilei" – Macerata

Il laboratorio di Fisica con Arduino

Introduzione ad Arduino e alla sua programmazione

Prof. Angelo Angeletti

19 gennaio - 1 febbraio 2018

Il progetto

Il Progetto Lauree Scientifiche (PLS) è attivo ormai da diversi anni e il Liceo Scientifico di Macerata, tranne in qualche rara occasione, ha sempre collaborato con gli altri licei della provincia che aderiscono a tale progetto e con il Dipartimento di fisica dell'Università di Camerino che coordina il tutto.

Anche per l'anno scolastico 2017-2018 si intende partecipare con una serie di conferenze e attività di laboratorio rivolte agli studenti di quarta e di quinta.

Ormai nel mondo della ricerca scientifica non si può prescindere dall'utilizzo di apparecchiature informatiche che permettono di acquisire i dati direttamente dall'esperimento. Non esiste più il ricercatore che pazientemente annota sul suo taccuino le misure che legge direttamente dagli strumenti. Sensori di alta precisione e computer ad essi collegati fanno il lavoro più noioso.

Lo scienziato deve però ancora progettare l'esperimento controllarne il funzionamento e poi interpretare i dati per ricavare una legge fisica o misurare il valore di una costante.

È in questa ottica che si vuol sviluppare il progetto per l'anno scolastico 2017 – 2018.

Una delle interfacce più popolari che si contraddistinguono per la semplicità di utilizzo è Arduino ed è proprio con questa che si intende implementare alcuni esperimenti di semplice svolgimento ma di sicura valenza didattica.

Il progetto

Obiettivi

L'obiettivo primario del progetto è quello di far lavorare gli studenti su esperimenti di un certo grado di complessità, ma comunque legati ai loro programmi di studi, partendo dal fenomeno e, attraverso la costruzione dell'apparato sperimentale (compreso il sistema di acquisizione dei dati), arrivare alla determinazione della legge fisica che descrive il problema o della misura di una grandezza.

Mezzi

Gran parte dei mezzi è già disponibile a scuola in quanto un progetto analogo è stato sviluppato un paio di anni fa riscuotendo notevole successo. Altri strumenti vengono messi a disposizione del dipartimento di fisica dell'Università di Camerino.

Crediti universitari

Agli studenti che parteciperanno al progetto e che poi si iscriveranno ad uno dei corsi di studio dell'Università di Camerino, saranno assegnati fino a quattro crediti formativi universitari, da utilizzare all'ambito di quelli riservati alle attività formative autonomamente scelte.

Le modalità di presentazione dei risultati e l'attribuzione dei crediti avverranno secondo la procedura prevista agli artt. 8 e 9 del ***“Regolamento di Ateneo per la realizzazione di progetti formativi tra UNICAM e gli Istituti di Istruzione Secondaria Superiore ai fini dell'attribuzione di Crediti Formativi Universitari”***.

Programma di massima

Fase 1 – Introduzione

1) – Venerdì 19 gennaio 2018

Prof. Angelo Angeletti: Presentazione del progetto. Un po' di teoria sulla scheda Arduino.

2) – Venerdì 26 gennaio 2018

Dr. Riccardo Natali (UNICAM): Elettronica analogica e digitale: la scheda Arduino

3) – Giovedì 1 febbraio 2018

Proff. Angelo Angeletti, Fabrizio Gentili: prime applicazioni con Arduino



Fase 2 – Laboratorio

4) – Venerdì 9 febbraio 2018

5) – Venerdì 16 febbraio 2018

6) – Venerdì 23 febbraio 2018

7) – Venerdì 2 marzo 2018

8) – Venerdì 9 marzo 2018

Fase 3 – Analisi dati e preparazione materiali finali

9) – Martedì 20 marzo 2018

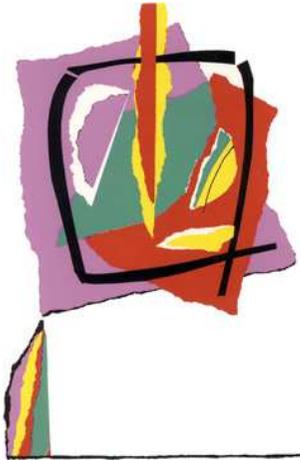
10) – Giovedì 22 marzo 2018

11) – Martedì 27 marzo 2018

Fase finale – Università di Camerino (?)

12) – Data da definire

Gli incontri si terranno presso i laboratori di fisica della sede centrale del Liceo Scientifico Galilei dalle 14.30 alle 16.30.



PROGETTO LAUREE SCIENTIFICHE FISICA

a.s. 2017-2018



Liceo Scientifico "G. Galilei" – Macerata

Il laboratorio di Fisica con Arduino
Introduzione ad Arduino e alla sua programmazione

Prof. Angelo Angeletti

19 gennaio 2018

Premessa

Per queste slide devo ringraziare FABIO CAPODAGLIO, uno studente di Fisica della laurea magistrale di UNICAM, che insieme al dott. Alessandro Saltarelli, nel 2015-2016, ha fatto anche la maggior parte del lavoro di preparazione di questo progetto.

Che cosa 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.



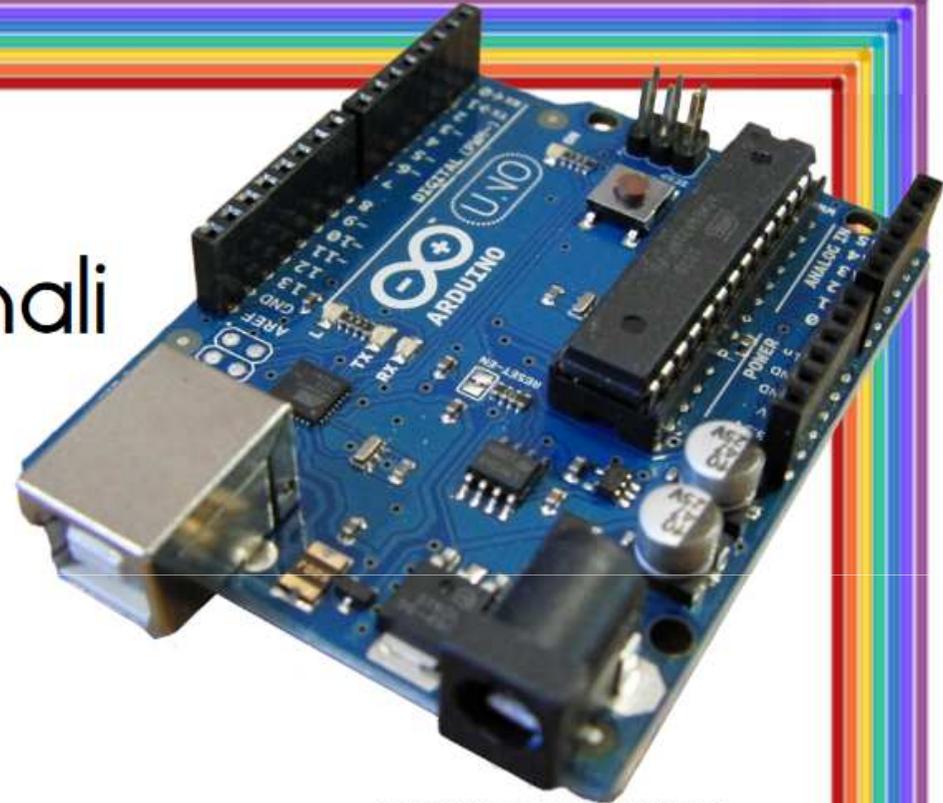
La scheda nasce nel 2005 grazie al lavoro fatto di Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis a Ivrea. Esistono diverse schede Arduino ma quella da noi usata è l'Arduino UNO REV 3 il cui processore è un ATmega328P. Tale scheda ha una memoria di 32Kb.

Che cosa è Arduino

Gli
Originali



Arduino mega



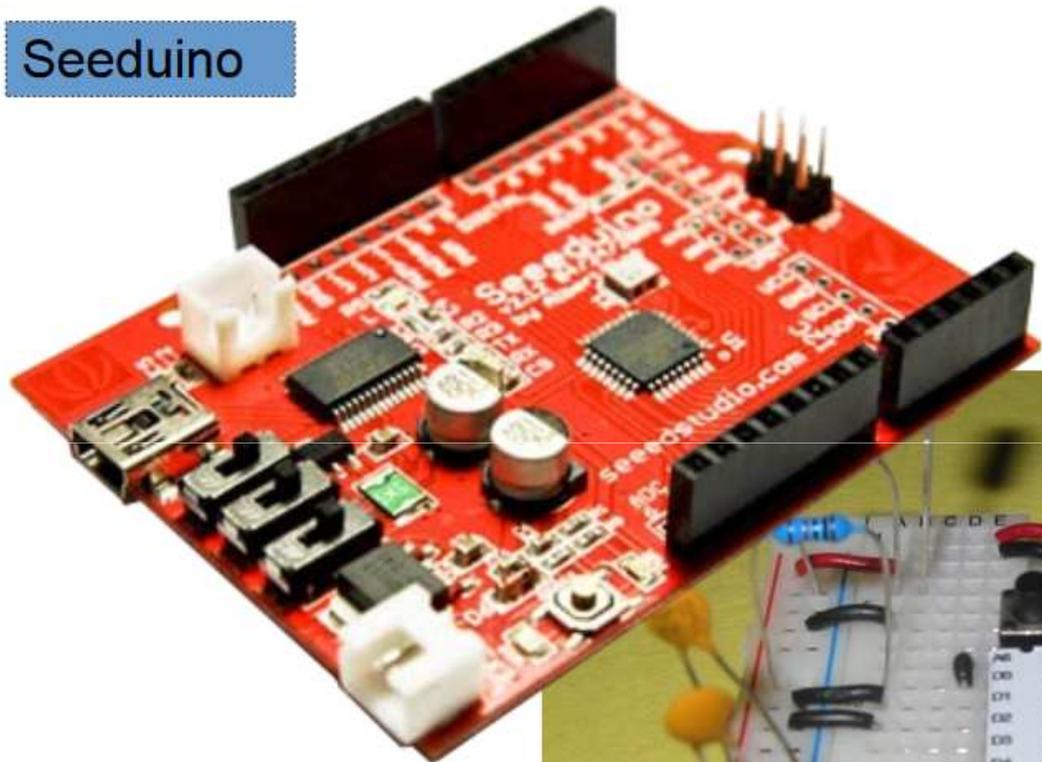
Arduino UNO



Arduino nano

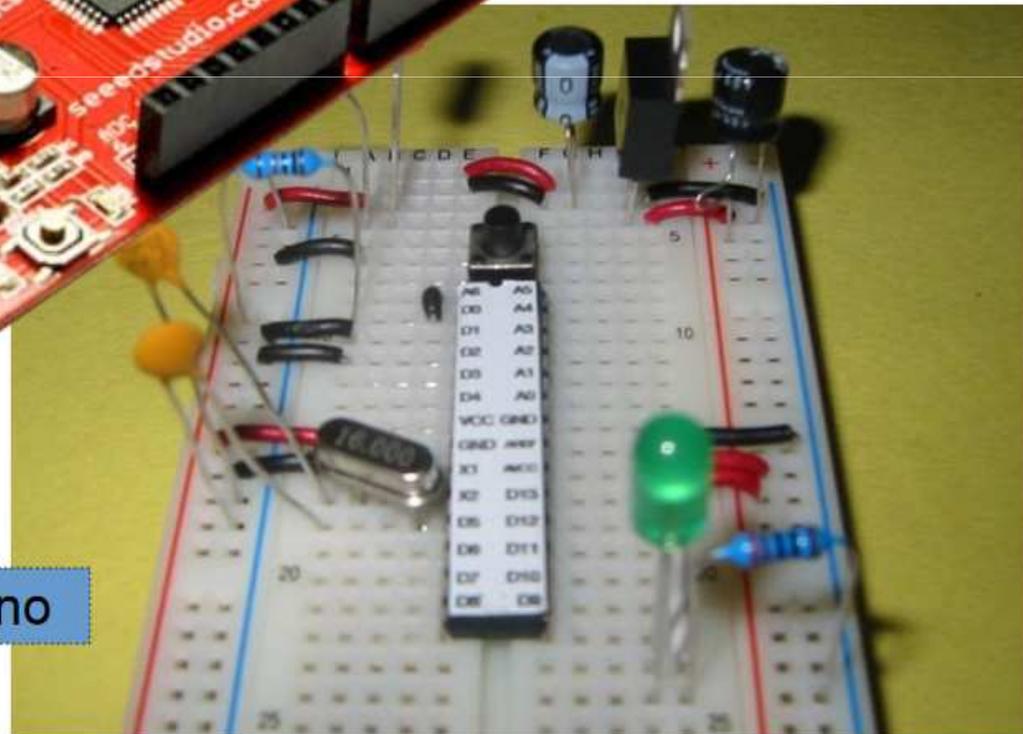
Che cosa è Arduino

Seeduino



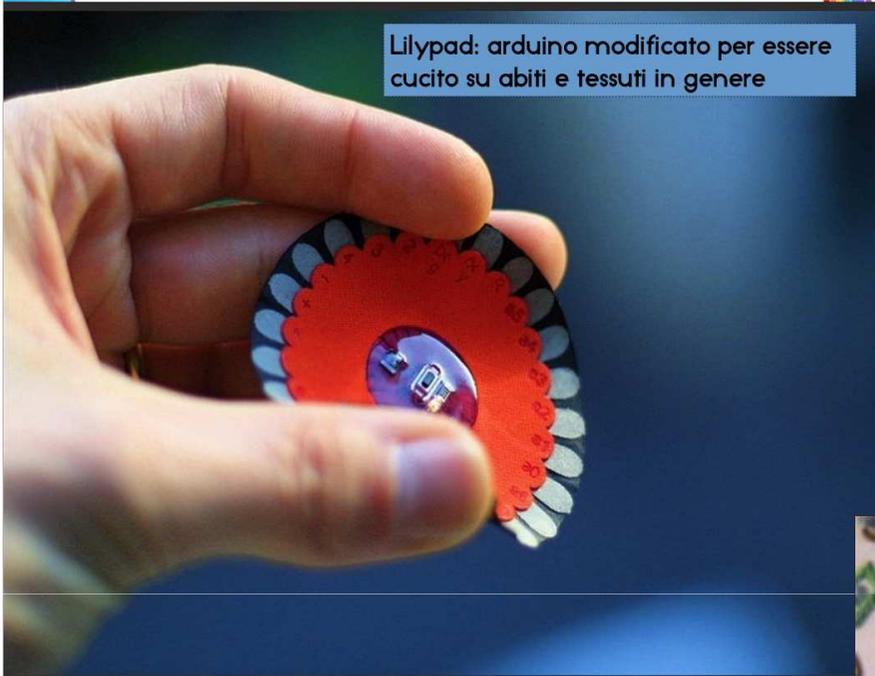
Cloni

BreadBoard Arduino



Che cosa è Arduino

LilyPad: arduino modificato per essere cucito su abiti e tessuti in genere



Applicazione di LilyPad su un tessuto

Che cosa è Arduino



Arduino Robot



Arduino Leonardo



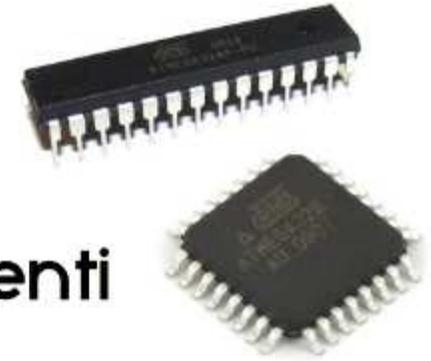
Arduino Yun



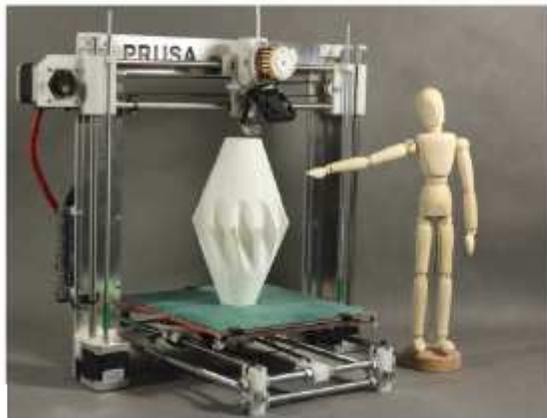
Arduino Esplora

Che cosa è un microcontrollore

Componente elettronico **programmabile** che consente di sviluppare dispositivi **intelligenti** a basso costo.



È un piccolo **computer**, con velocità e memoria ridotta, quindi consumi inferiori



Che cosa ci fai con Arduino

TANTE COSE

Noi apparati per acquisizione dati in laboratorio di fisica.

In internet esistono centinaia di progetti.

Dove lo troviamo

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

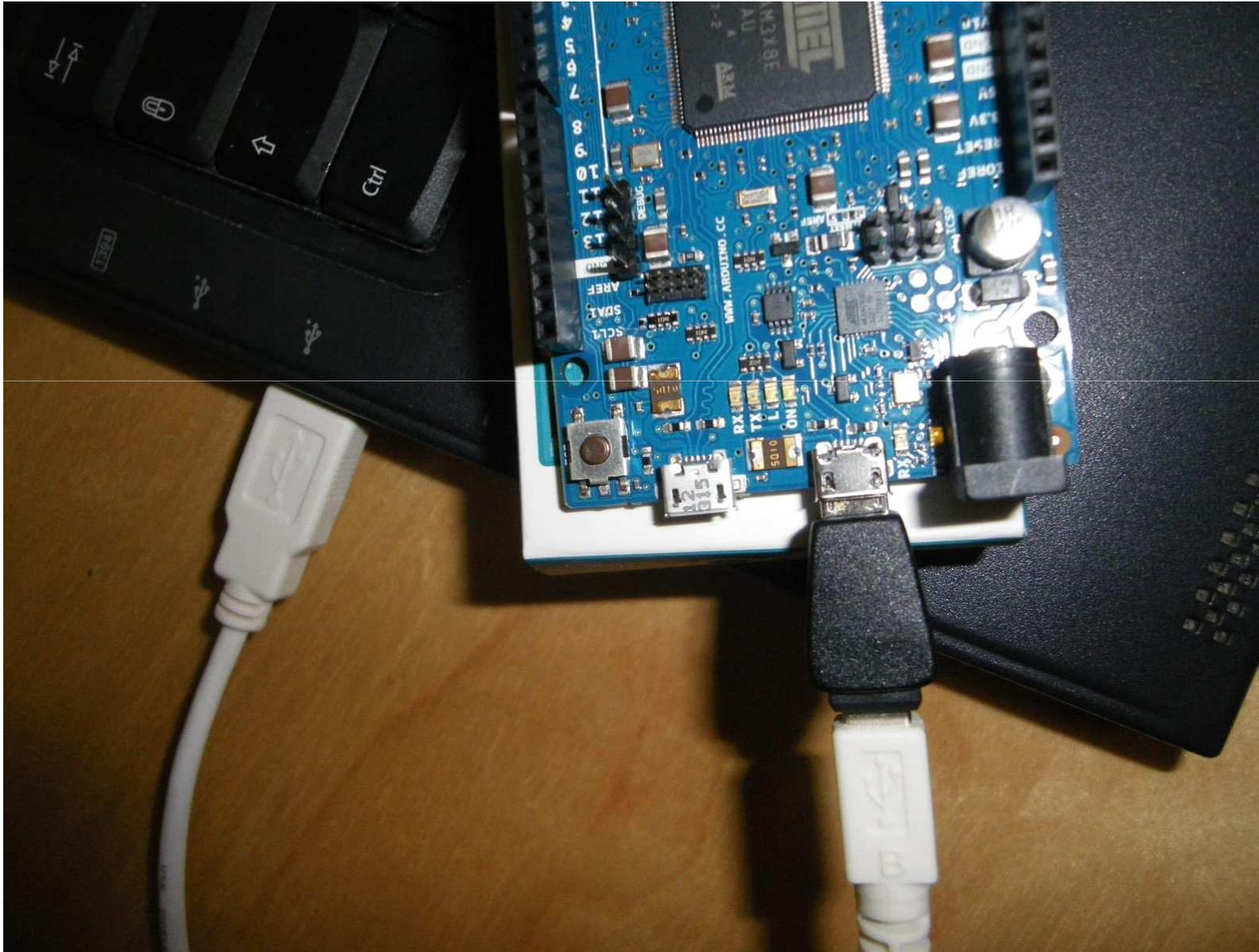
<http://www.html.it/guide/guida-arduino/>

<http://www.mauroalfieri.it/corso-arduino-on-line.html>

<http://share.dschola.it/castigliano/elettronici/5f/Materiali/ARDUINO/Tiziana Marsella - Programmare Arduino.pdf>

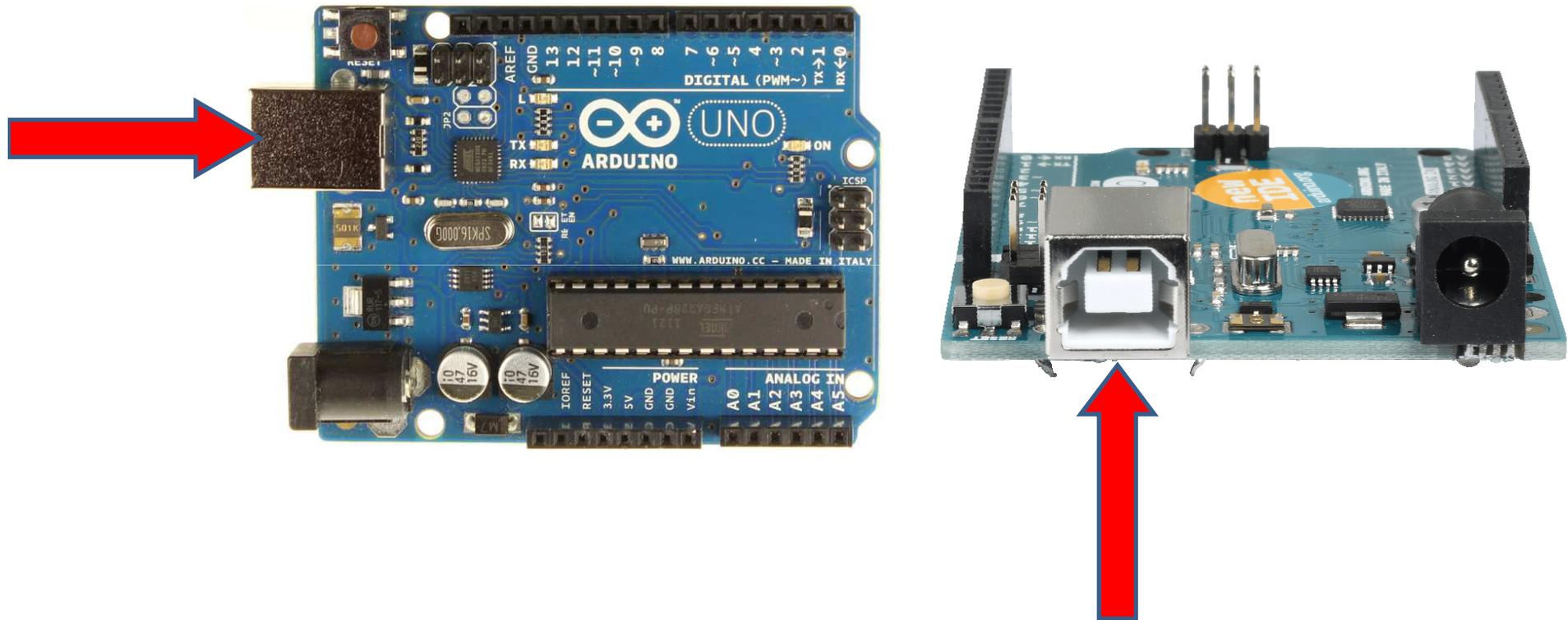
Incominciamo

La scheda Arduino va collegata al computer tramite un cavetto USB.



Alimentazione della scheda

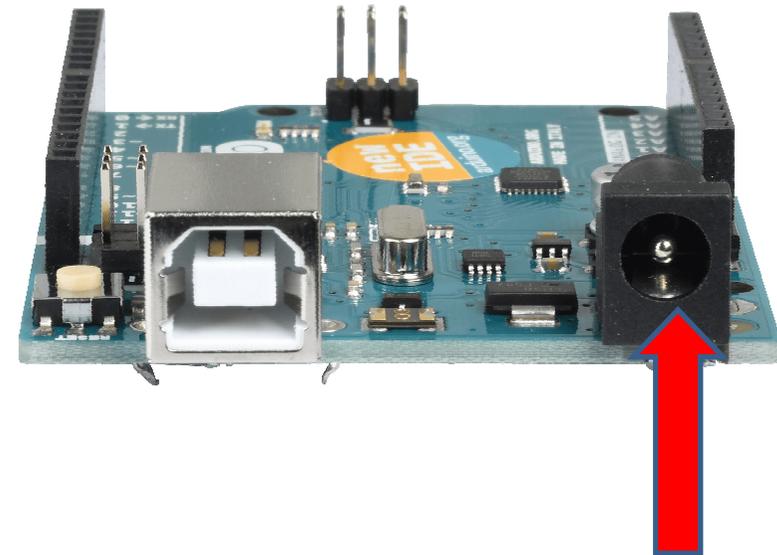
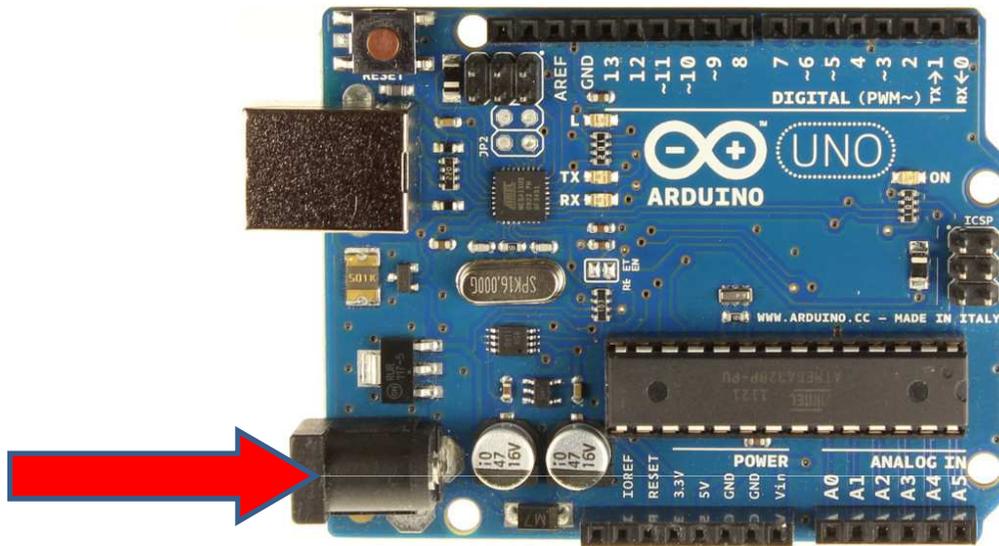
METODO 1: Collegando la scheda alla porta USB del computer



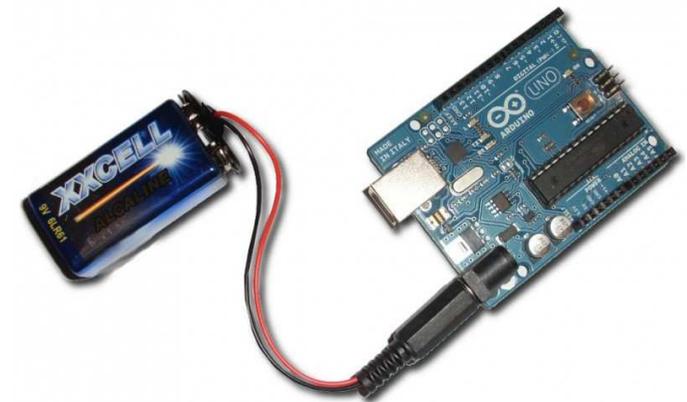
La scheda è alimentata con una tensione di 5 V e una corrente di 500 mA (valori ideali di funzionamento del processore).

Alimentazione della scheda

METODO 2: Collegando il connettore di alimentazione

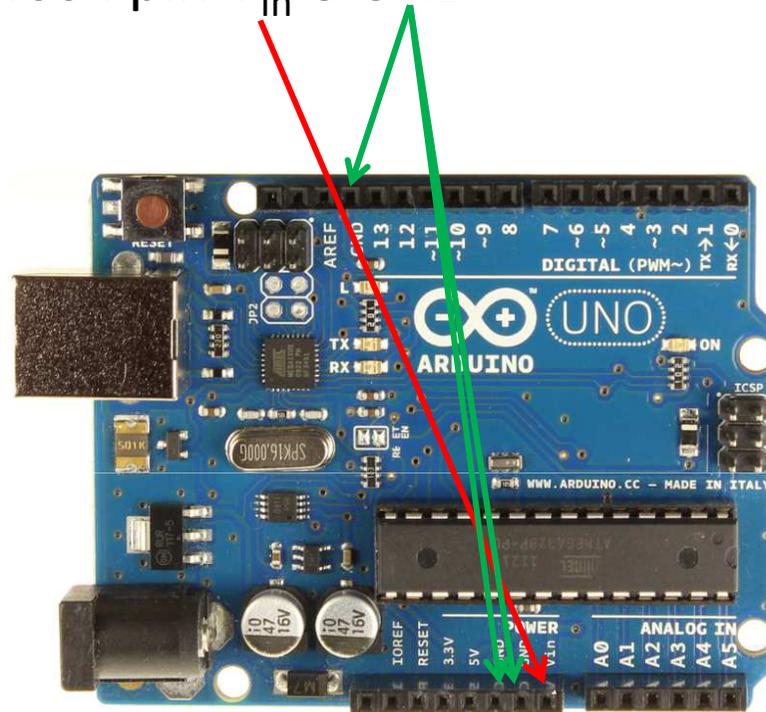


Al connettore va collegata una tensione compresa tra i 7 V e i 12 V. Un regolatore di tensione montato sulla scheda porta tale tensione ai valori ideali; in questo caso la corrente può arrivare a 800 mA. Sul connettore è presente una protezione per eventuali inversioni della polarità.



Alimentazione della scheda

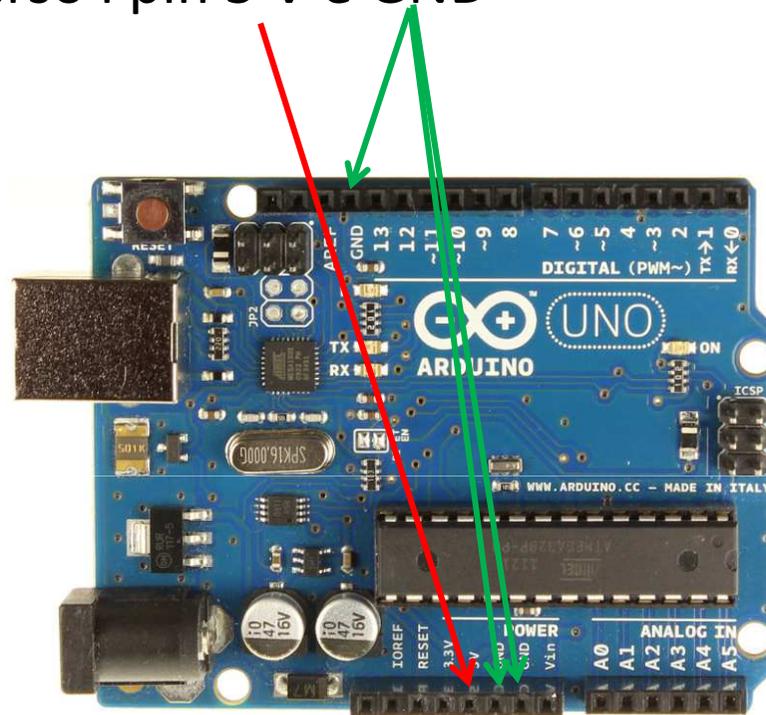
METODO 3: Attraverso i pin V_{in} e GND



In questo caso dobbiamo collegare una tensione compresa tra 6.5 V e 12 V (il polo positivo va collegato sul V_{in} e il polo negativo sul GND). Tale tensione passa attraverso il regolatore di tensione ma in questo caso non c'è protezione quindi bisogna fare attenzione a non invertire le polarità per non bruciare la scheda!!!

Alimentazione della scheda

METODO 4: Attraverso i pin 5 V e GND

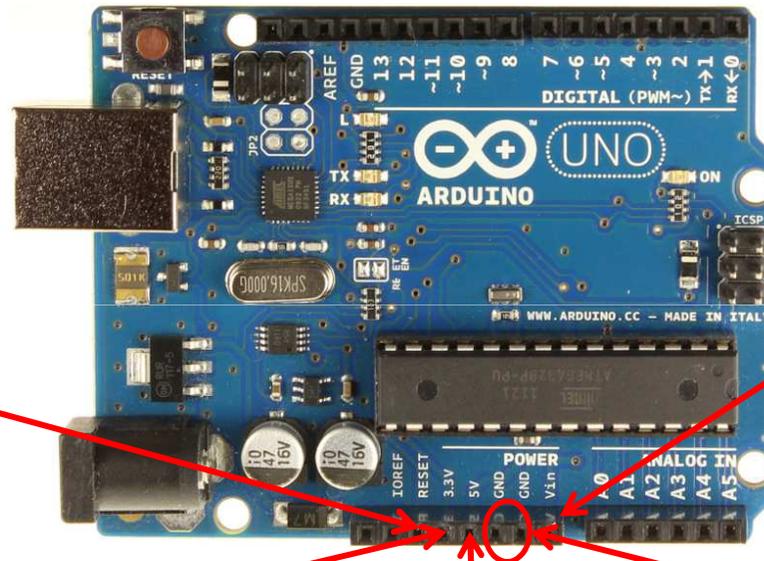


In questo caso dobbiamo collegare una tensione compresa tra 4.5 V e 5.5 V (il polo positivo va collegato sul 5 V e il polo negativo sul GND).

In questo caso non si ha né il regolatore di tensione né la protezione quindi se si superano i 5.5 V oppure si invertono le polarità si rompe la scheda.

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)



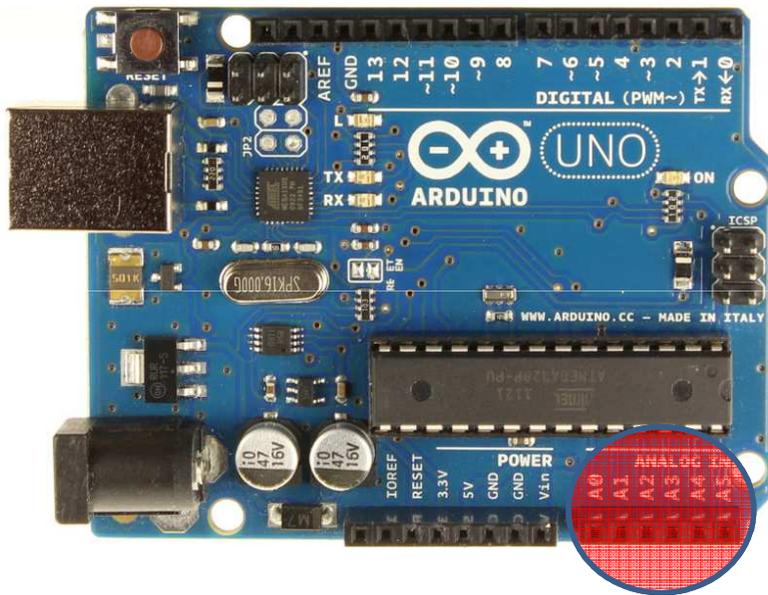
pin Vin: si può usare per alimentare la scheda

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:

$$2^{10} = 1024.$$

Pin analogici

Gli ADC ci permettono di leggere una tensione nel range 0 – 5 V.

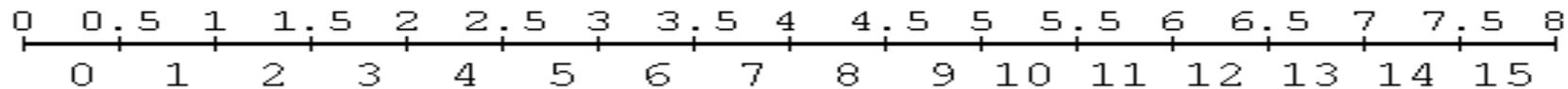
Tuttavia il comando di Arduino ***analogRead***, che permette di fare questa operazione, ci dà il risultato della misura in canali (quindi un valore compreso fra 0 e 1023).

Per convertire il valore misurato in volt possiamo usare la seguente

relazione:

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

Convertitore analogico-digitale



Supponiamo di avere un ADC con risoluzione di 4 bit.

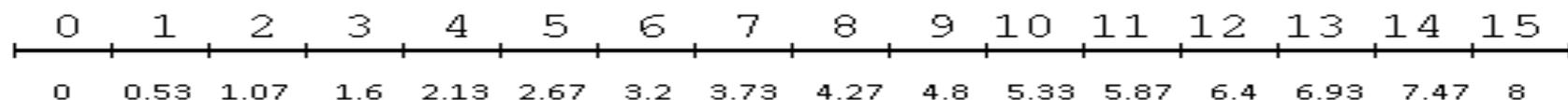
Significa che possiamo avere 16 canali ($C = 2^4 = 16$) numerati da 0000 (= 0) fino a 1111 (=15), ossia C assume tutti i valori naturali da 0 a 15.

Supponiamo ora di dover trasformare una grandezza X che può assumere valori da 0 fino a 8.

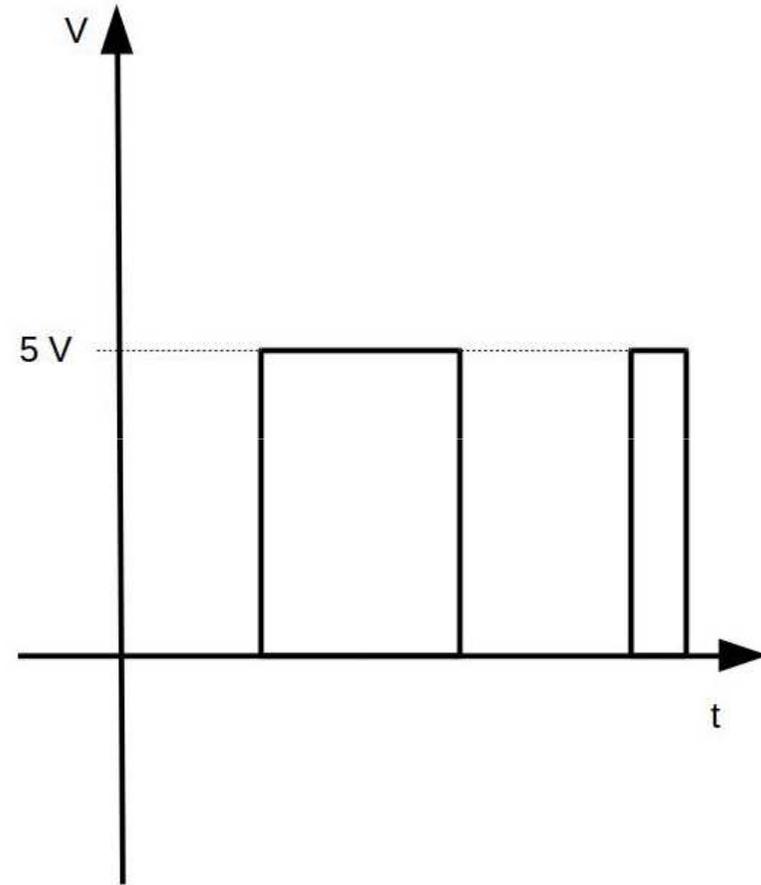
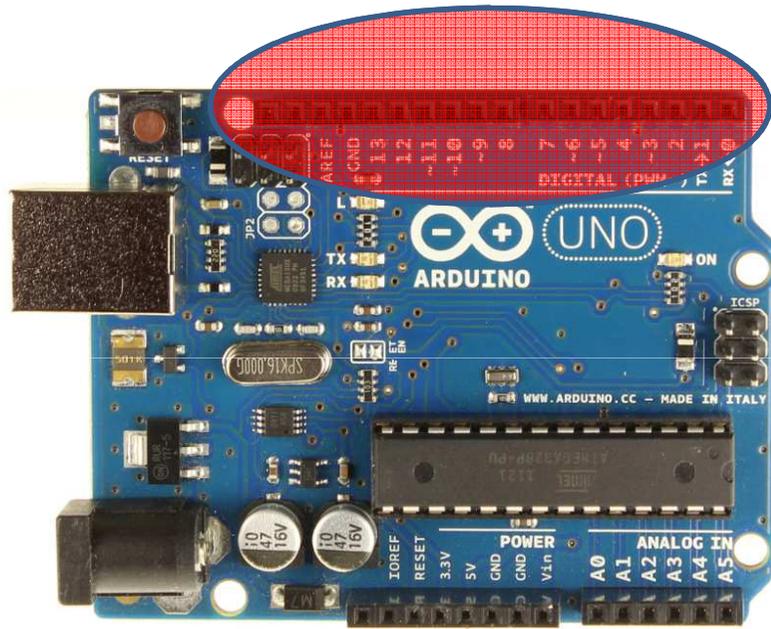
$C = 0 \Rightarrow X$ compreso tra 0 e 0.5, $C = 1$ significa X compreso tra 0.5 e 1 e così via fino al $C = 15$ che significa X compreso tra 7.5 e 8.

Ad esempio se la grandezza $X = 3,25$ avremo $C = 6$.

Se invece abbiamo $C = 6$ allora $X = 6 \cdot (8/15) = 3,2$.



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

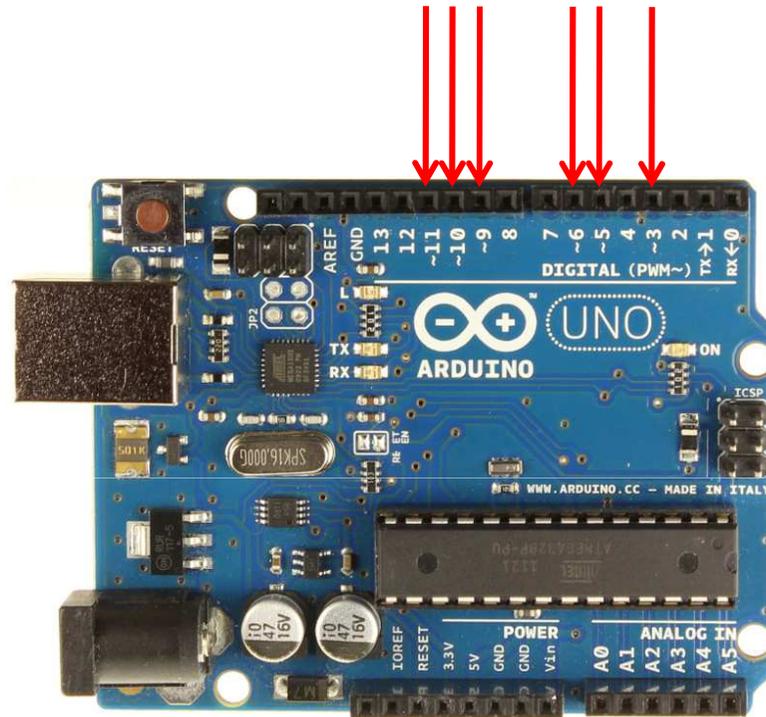
Quando i pin digitali vengono impostati come input il loro stato può essere:

LOW: se riceve in ingresso un segnale di circa 0 V

HIGH: se riceve in ingresso un segnale di circa 5 V

Come vedremo il sensore di posizione a ultrasuoni hr04, che useremo in alcuni esperimenti, utilizzerà due pin digitali, uno in input e uno in output.

Pin digitali speciali \Rightarrow 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)

Pin digitali speciali \Rightarrow Pulse Width Modulation (PWM)

I pin PWM sono dei pin digitali che possono fornire in output una tensione variabile da 0 V a 5 V.

Analogamente al caso della lettura della tensione con le porte analogiche (per la lettura si usava ***analogRead***) 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:

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

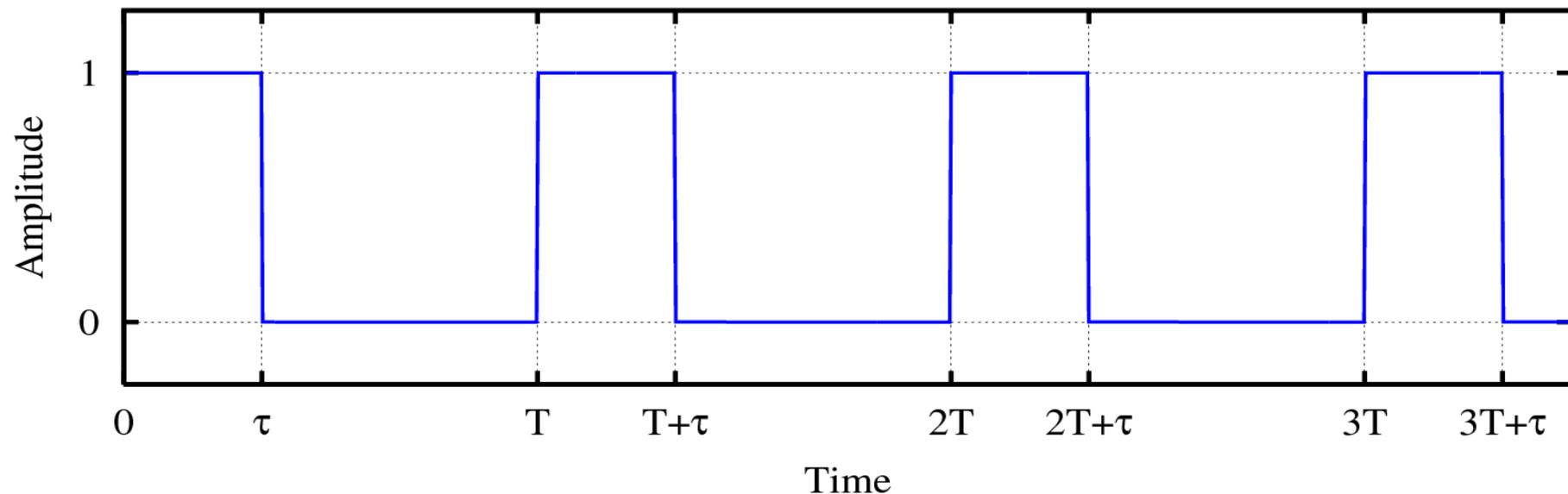
Pin digitali speciali \Rightarrow Pulse Width Modulation (PWM)

La tensione viene modulata attraverso il metodo della Pulse Width Modulation

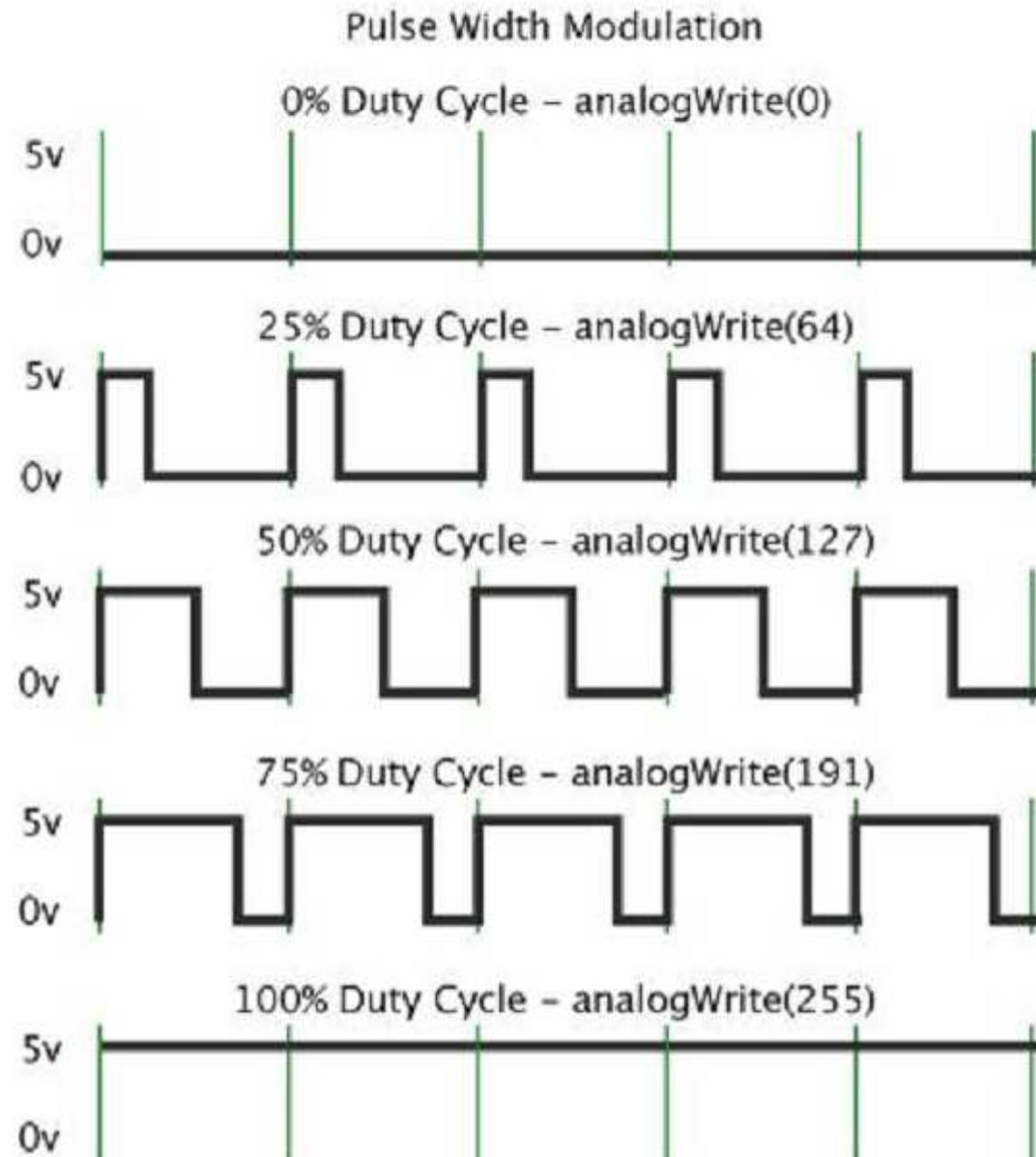
(modulazione di larghezza di impulso).

Esso consiste in una serie di impulsi ad una determinata frequenza (e quindi ad un determinato periodo T) che restano accesi per un tempo $\tau \leq T$.

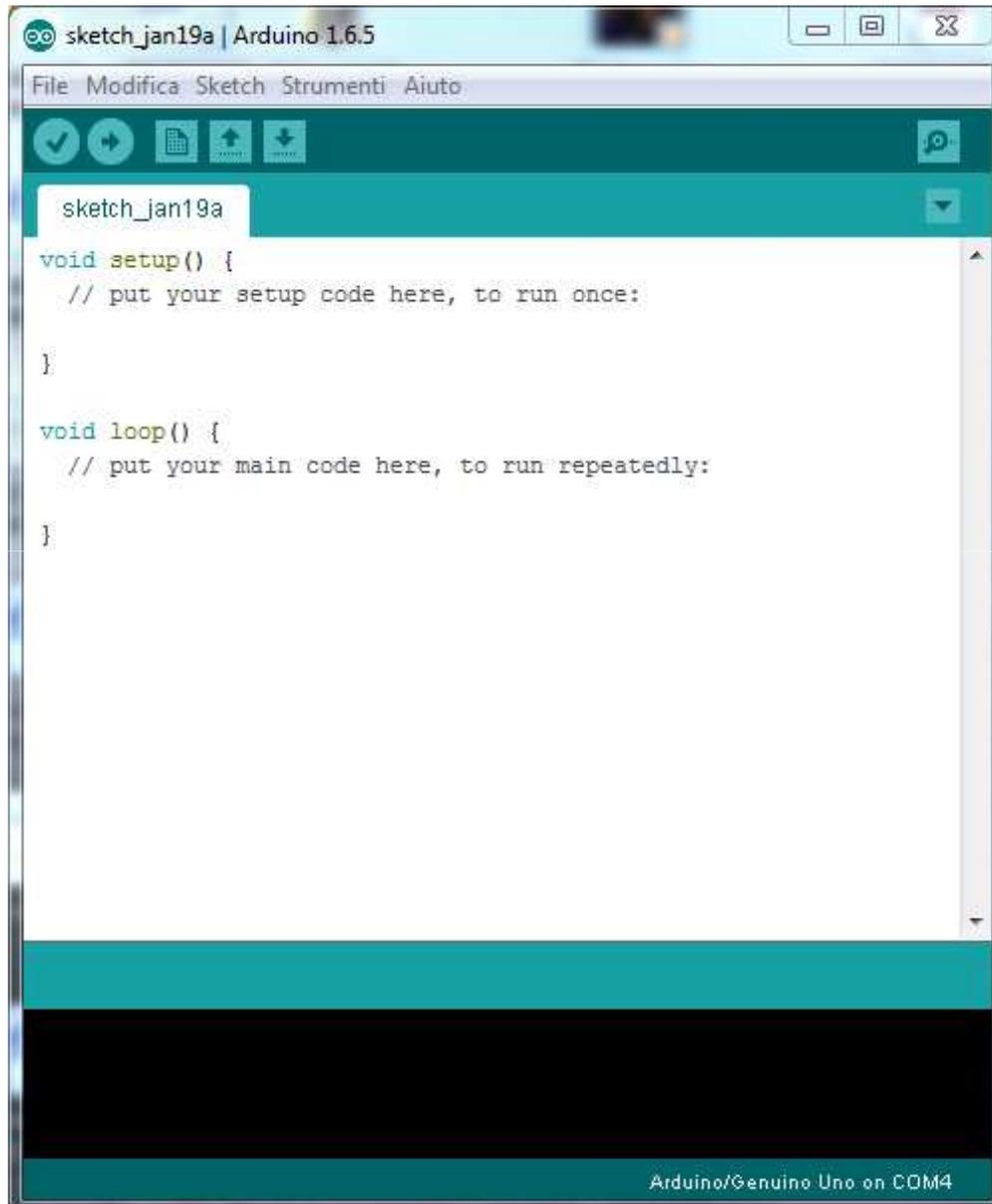
In base al valore di τ abbiamo quindi un diverso valore della tensione in uscita.



Pin digitali speciali \Rightarrow Pulse Width Modulation (PWM)



Come programmare Arduino

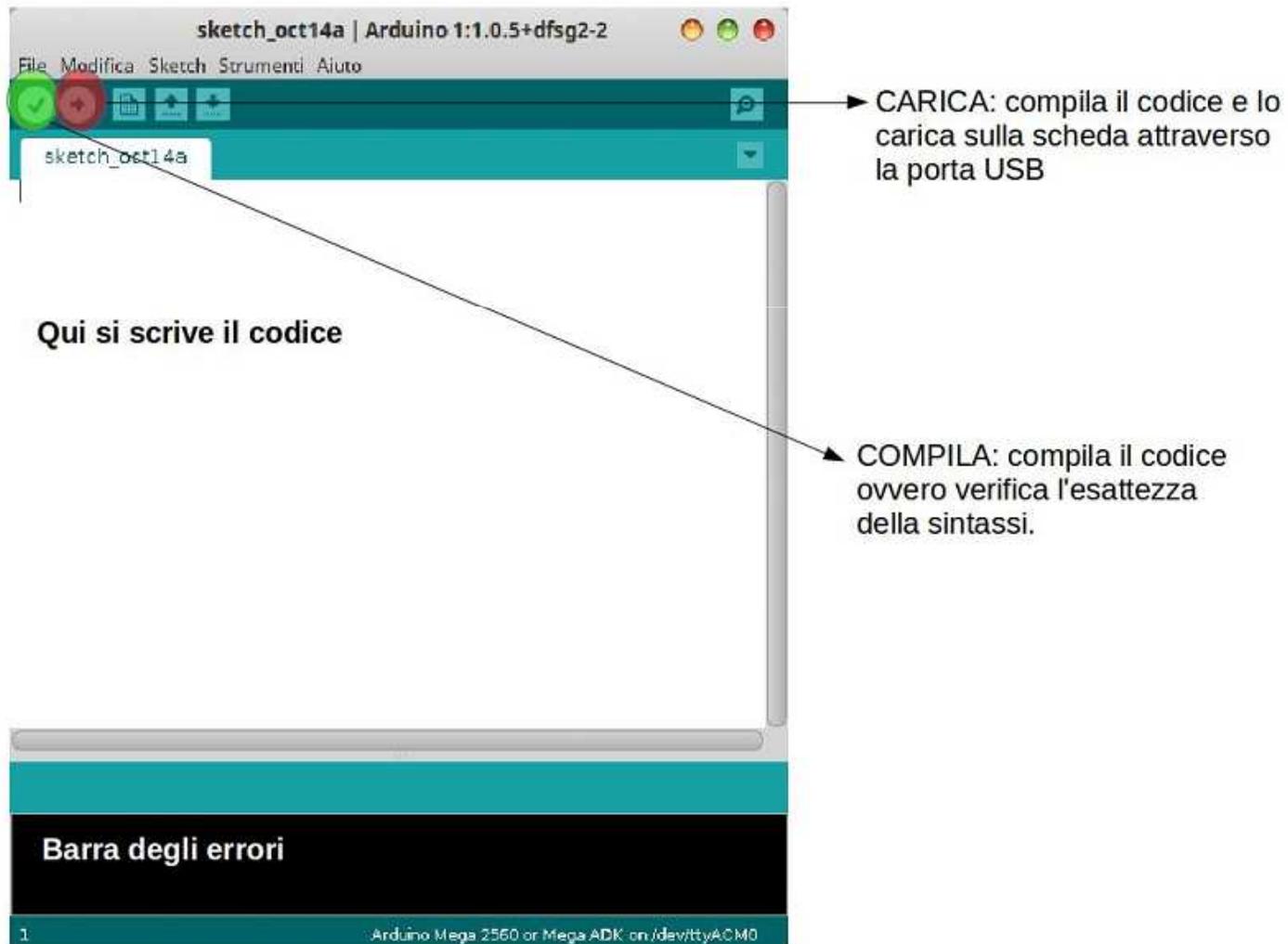


Per sviluppare il software per far funzionare la scheda Arduino, si utilizza ***l'ambiente di sviluppo (IDE)*** che ci permette di scrivere, compilare e trasferire i nostri programmi sulla scheda.

<https://www.arduino.cc/en/Main/Software>

Come programmare Arduino

La scheda Arduino viene programmata in un linguaggio molto simile al C utilizzando l'Arduino ide.



Come programmare Arduino

Una volta aperto il programma si può iniziare a scrivere il codice nell'apposito spazio.

Quando il codice (o parte di esso) è stato scritto per verificarne la correttezza bisogna innanzitutto salvare il file.

Fatto questo si può cliccare sul pulsante compila. 

Se la compilazione non va a buon fine significa che c'è qualche errore nella sintassi e tali errori sono riportati (in maniera più o meno chiara) nella barra degli errori.

Una volta che la compilazione va a buon fine e che l'intero codice è stato scritto si procede con il caricamento del programma (in gergo **sketch**) sulla scheda cliccando sul pulsante carica. 

Come programmare Arduino

Che cosa fare se la compilazione va a buon fine ma il caricamento sulla scheda 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!!!

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 tre modi:

- Spegnere e riaccendere Arduino togliendo e rimettendo l'alimentazione alla scheda.
- Premere il tasto reset.
- Utilizzare il software freeware (solo per Windows) Gobetwino.

Come usare Gobetwino

Il programma Gobetwino, una volta lanciato, esegue il reset della scheda e fa ripartire lo sketch dall'inizio.

Questo software permette inoltre di definire un comando per la stampa su file.

Metodo alternativo per lanciare una misura e stampare i dati!

Un metodo alternativo all'utilizzo di Gobetwino è quello di utilizzare il **monitor seriale** di Arduino (menù *strumenti* → *monitor seriale*).

Quando si apre il monitor seriale Arduino inizia di nuovo l'esecuzione dello sketch e stampa i dati sul monitor.

Tuttavia si hanno due grossi svantaggi:

- 1) Non si ha la possibilità di esportare direttamente i dati su un file. L'unico metodo è il copia e incolla che non è ideale quando si devono selezionare grandi quantità di dati.
- 2) Se i dati sono davvero molti non si ha la possibilità di vederli tutti sul monitor.

Ma noi utilizzeremo questo metodo.

Struttura di uno *sketch* per Arduino

```
Blink | Arduino 1:1.0.5+dfsg2-2
File Modifica Sketch Strumenti Aiuto
Blink §
#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:

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 §
#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.
```

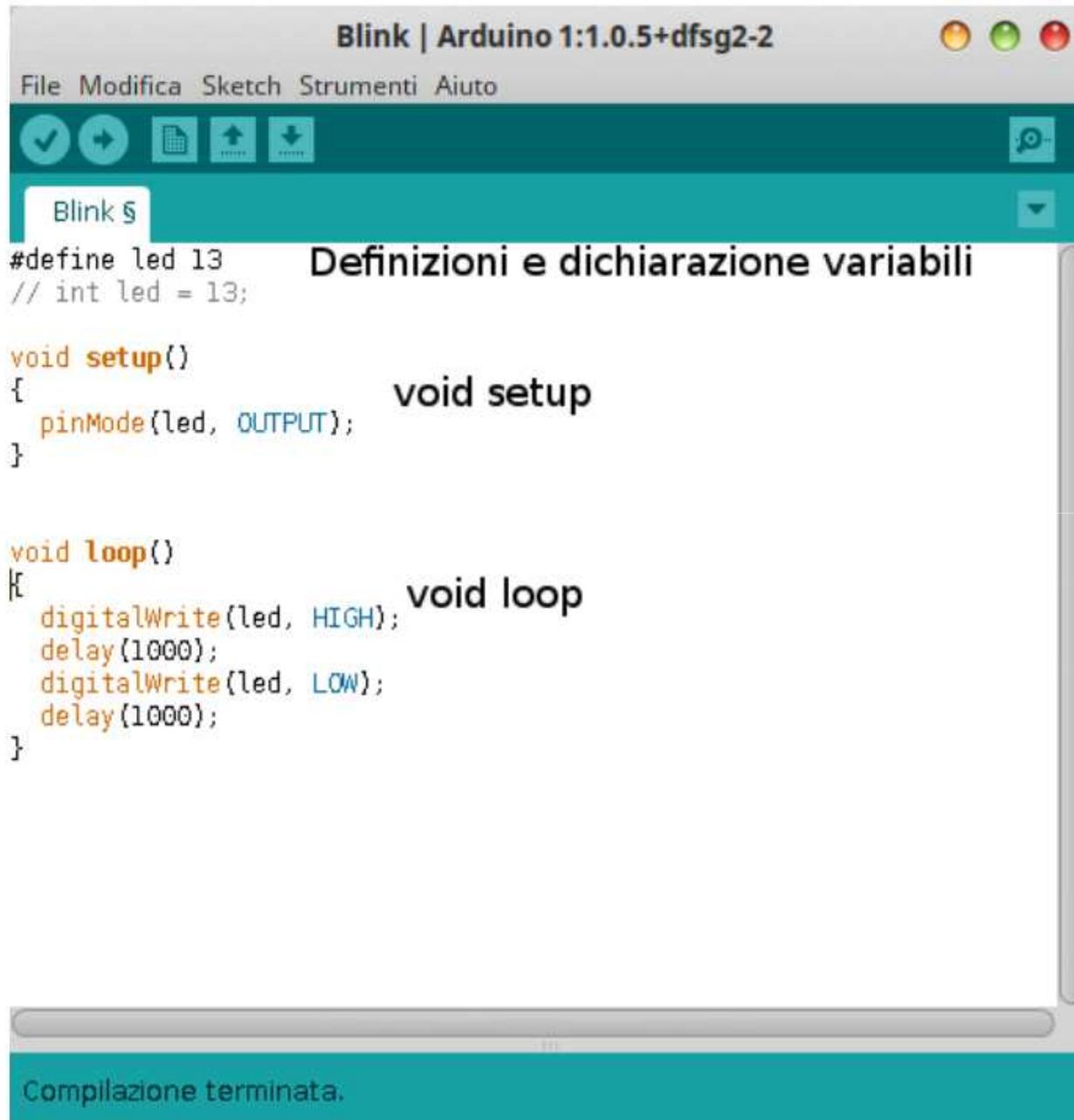
Uno sketch per Arduino si divide in tre parti:

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.

Importazione delle librerie

A volte, soprattutto quando si utilizzano dei sensori e degli shield (cioè dei moduli che si applicano sulla scheda Arduino) potrebbe essere necessario importare delle librerie esterne.

Il modo più semplice per fare ciò è avere il file .zip di tale libreria ed importarla dal menù *sketch* → *aggiungi libreria*.

Una volta aperto il menù bisogna selezionare la libreria che si vuole importare nella cartella dove è contenuta.

Una volta importata la libreria (tale operazione va fatta una sola volta in quanto una volta fatta la libreria rimane nella cartella libraries di Arduino) bisogna richiamarla all'inizio dello sketch utilizzando il seguente comando:

#include <nomelibreria.h> (ad esempio **#include <OneWire.h>**)



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*.

Collegamenti in angeloangeletti.it

Definizione e dichiarazione variabili

Definizioni: si ha la possibilità di assegnare un nome ai pin di Arduino 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 (infatti i comandi per le porte analogiche sono diversi da quelli per quelle digitali).

Definizione e dichiarazione 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).

Esistono anche altri tipi di variabili come ad esempio interi senza segno, costanti, ecc.

Definizione e dichiarazione variabili

Le variabili oltre ad essere dichiarate possono anche essere inizializzate ad un valore iniziale. Ad esempio:

```
int pippo;
```

```
int pippo1 = 0;
```

```
double misura;
```

In pratica la struttura è la seguente:

```
tipo di variabile nome variabile = valore iniziale ;
```

ATTENZIONE al punto e virgola: come nel linguaggio C il punto e virgola va messo alla fine di ogni istruzione (sono escluse le importazioni delle librerie e le definizioni).

Definizione di vettori

Oltre alle variabili scalari si possono definire anche dei vettori (o meglio delle liste di numeri → *array*).

I numeri contenuti in tali liste sono tutti dello stesso tipo (o tutti interi o tutti in virgola mobile).

Per definire un vettore si usa la seguente sintassi:

tipo nomevariabile[numero elementi vettore]

int vettore[100];

Possiamo allocare il vettore (specificarne in numero di elementi) anche dinamicamente ovvero:

int Nmis = 100;

double vettore[Nmis];

Tuttavia in questo caso il vettore va definito dentro il void loop.

Che 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.

Che cosa mettere nel *void setup*

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 loop***); il comando è:

digitalWrite(nomeporta / numeroporta, HIGH / LOW)

Tale comando serve a scrivere HIGH o LOW sulla porta digitale specificata.

Che 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.

RICORDA: vanno messi nel ***void loop*** anche le dichiarazioni dei vettori allocati dinamicamente ovvero quei vettori il cui numero di elementi non è specificato direttamente da un intero ma da una variabile intera.

Nelle prossime slides andremo a vedere sia la sintassi dei comandi di Arduino per il ***void loop*** sia quella dei cicli.

Comandi di Arduino per il *void loop*: 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 molto utile con il sensore a ultrasuoni che useremo negli esperimenti.

Comandi di Arduino per il *void loop*: porte digitali

analogWrite(nomeporta / numeroporta, 0-255)

Comando per scrivere su una porta digitale PWM

Il valore 0 corrisponde a 0 V mentre il valore 255
corrisponde a 5 V

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)

Come fare una misura di tensione con Arduino

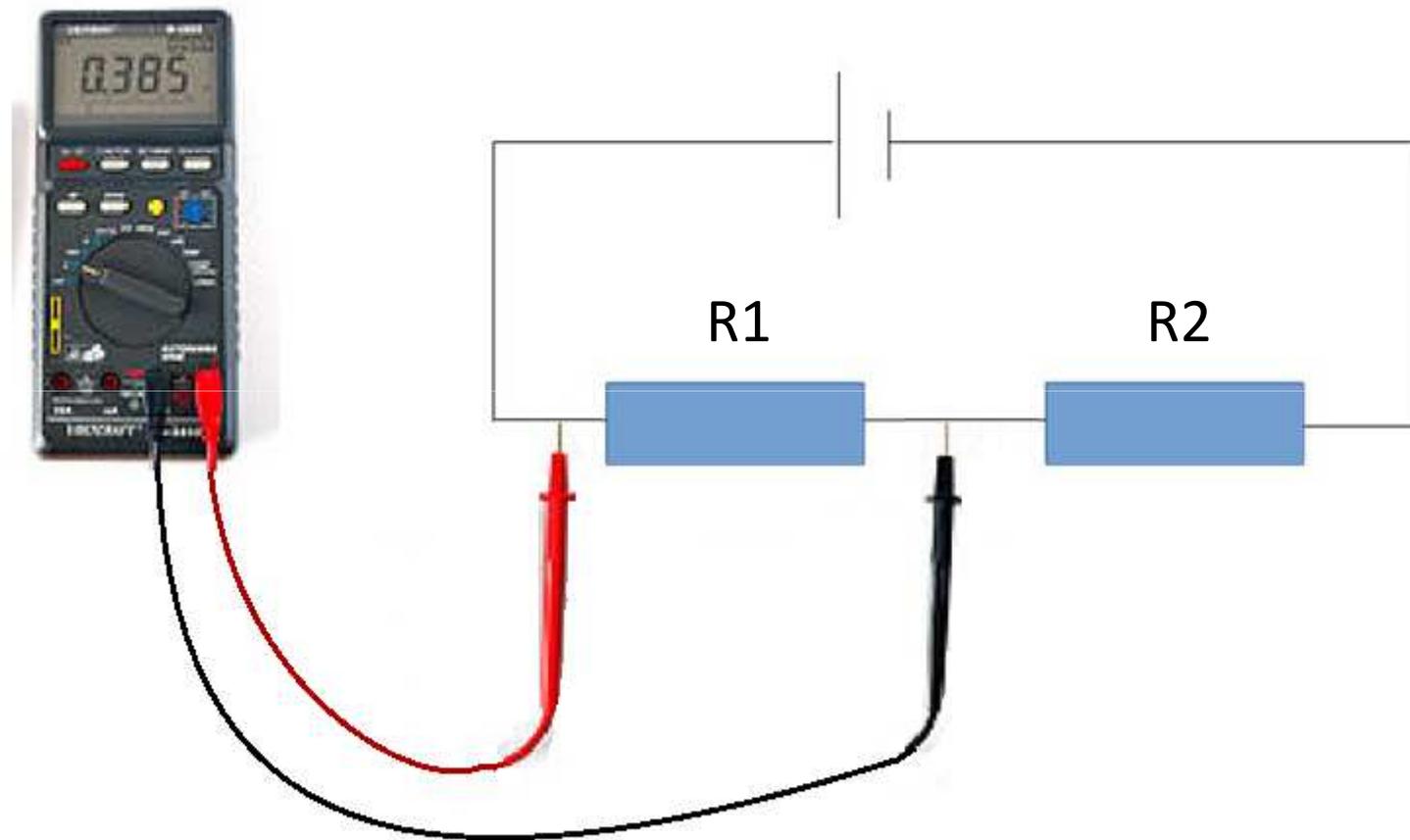
RICORDA

Quando si vuole fare una misura di tensione ai capi di un dispositivo (ad esempio una resistenza) si deve immaginare di usare Arduino come un multimetro; quindi i cavi da collegare sono due (il puntale rosso e quello nero del multimetro).

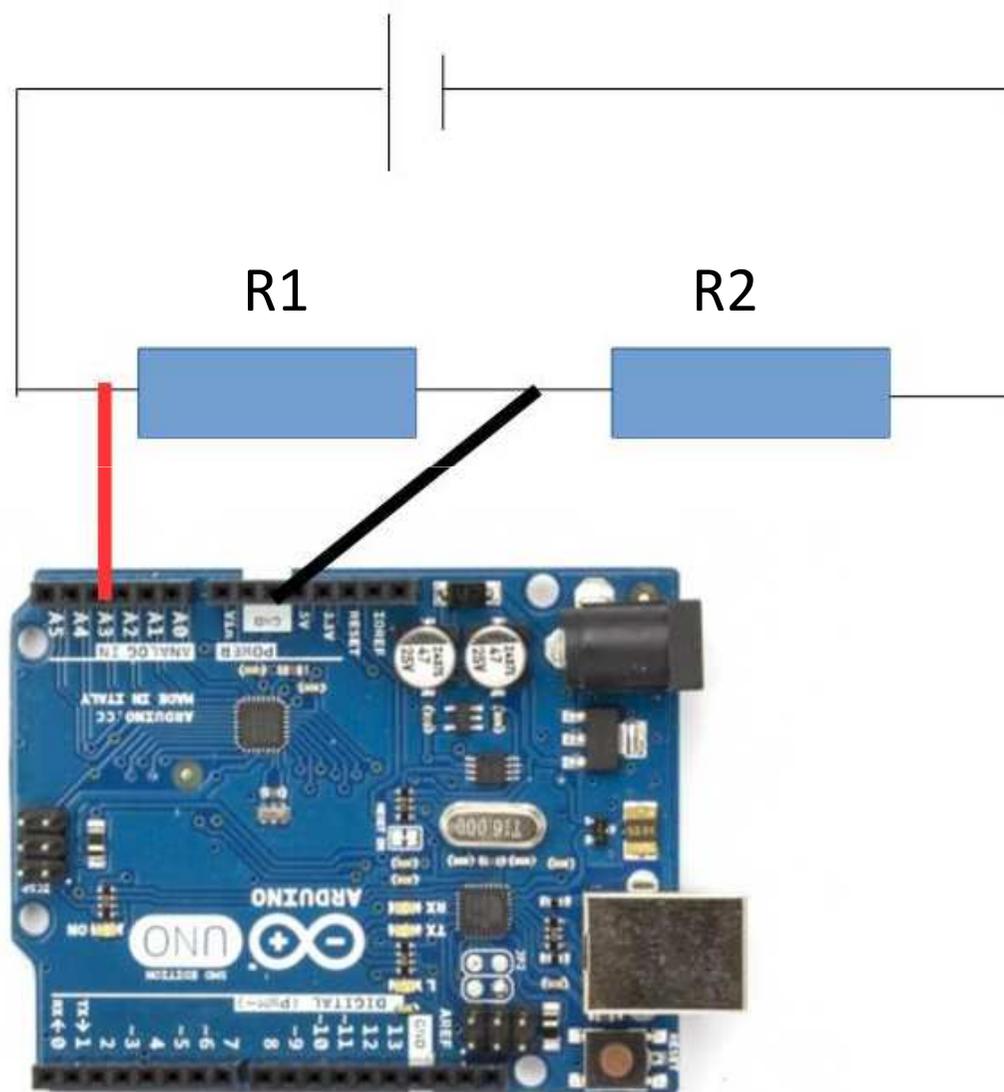
Il puntale rosso è la nostra porta analogica mentre il puntale nero è una delle porte di ground (GND) di Arduino.

Attenzione a non invertire perché Arduino legge tensioni tra 0 e 5 V quindi non è in grado di leggere tensioni negative e se invertiamo rosso e nero (più e meno) in realtà abbiamo una tensione negativa ma in pratica Arduino legge 0 V.

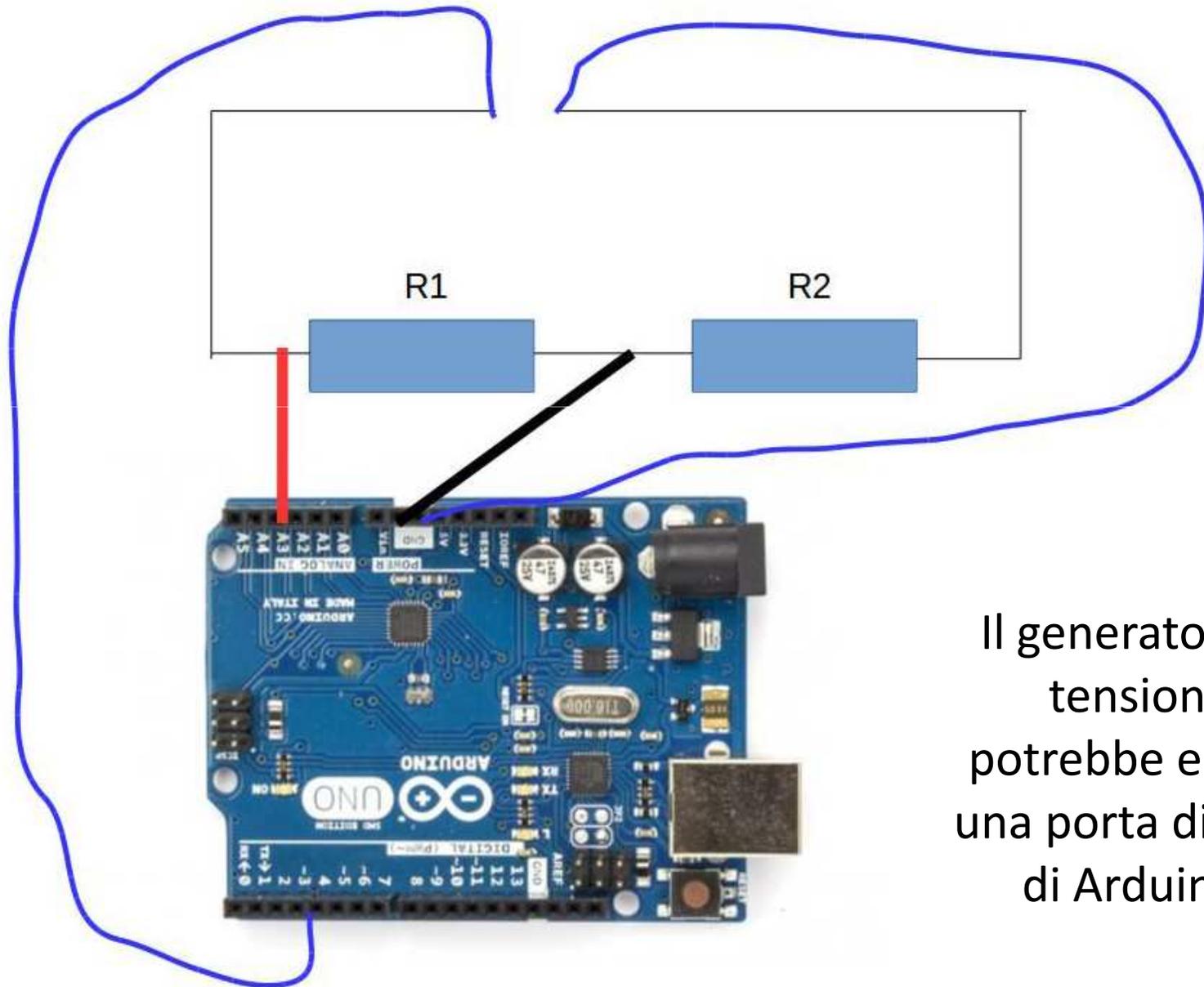
Come fare una misura di tensione con Arduino



Come fare una misura di tensione con Arduino

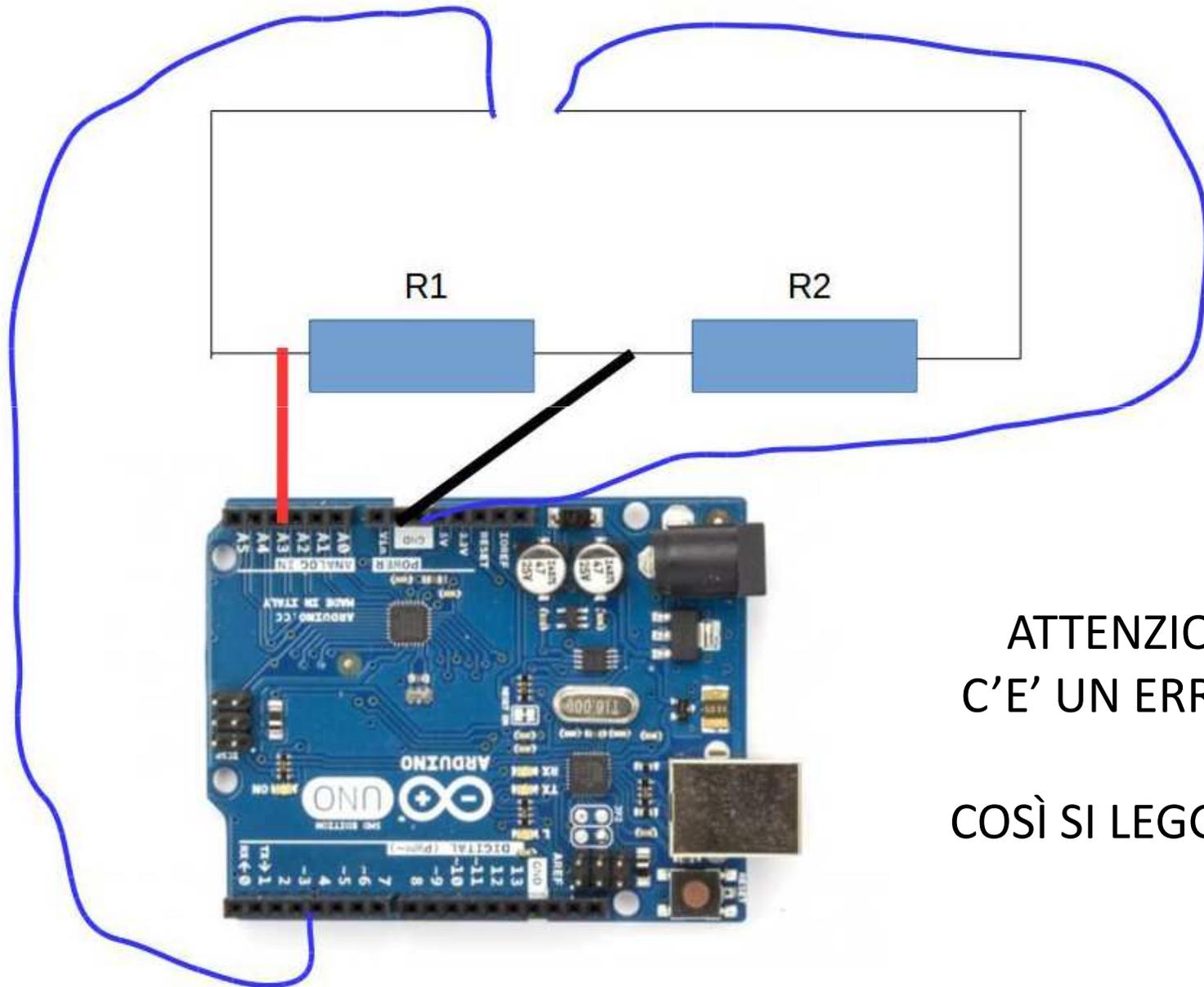


Come fare una misura di tensione con Arduino



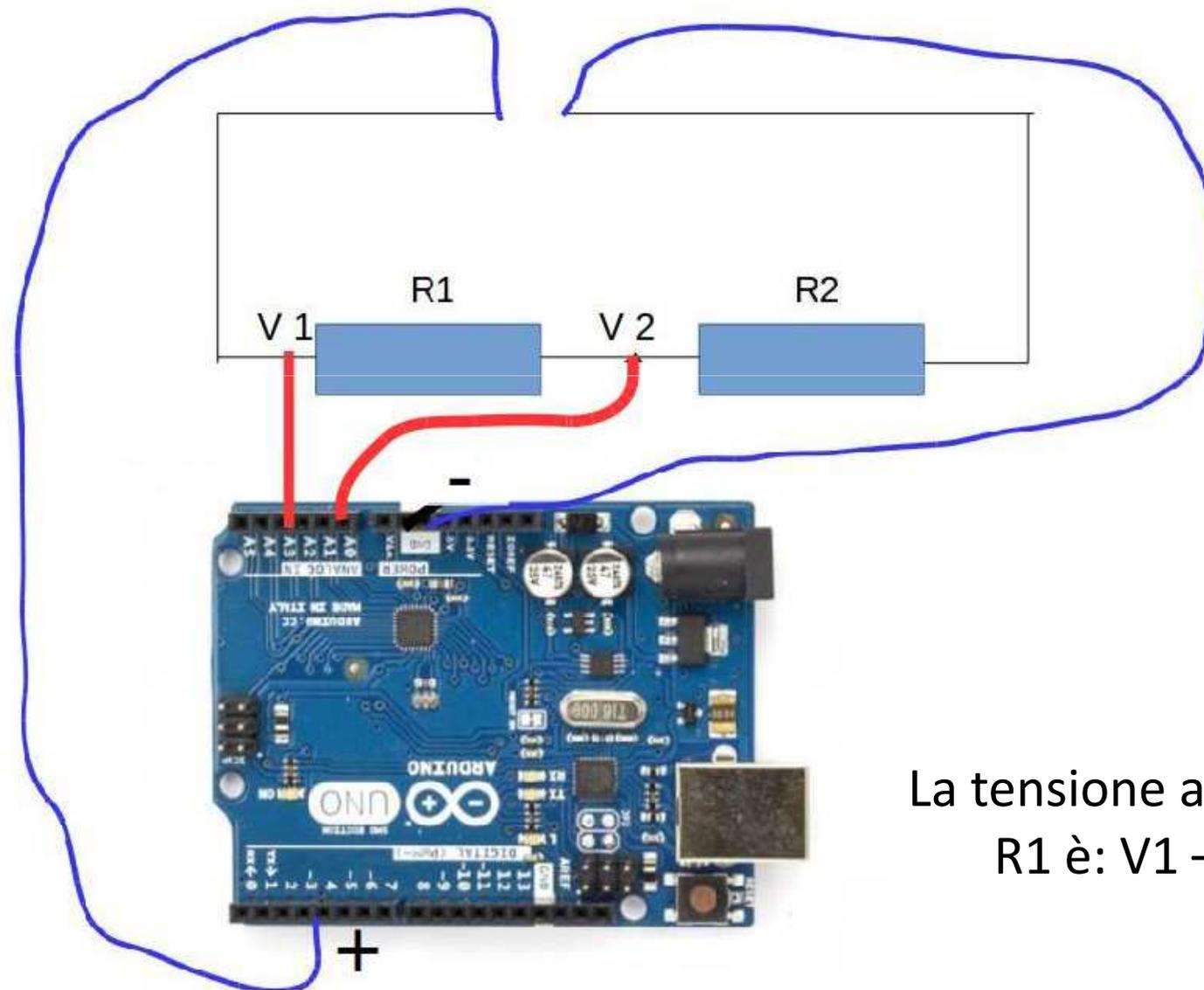
Il generatore di tensione potrebbe essere una porta digitale di Arduino!

Come fare una misura di tensione con Arduino



Come fare una misura di tensione con Arduino

ECCO IL METODO CORRETTO!



La tensione ai capi di
R1 è: $V1 - V2$

Altre funzioni nel *void loop*

Un altro tipo di funzioni che incontreremo spesso sono quelle che permettono di inserire dei ritardi di tempo (o meglio dei tempi di attesa) tra un'istruzione e l'altra e quelle che permettono di contare il tempo trascorso da quando è stato lanciato lo sketch.

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)

Altre funzioni nel *void loop*

Un altro tipo di funzioni che incontreremo spesso sono quelle che permettono di inserire dei ritardi di tempo (o meglio dei tempi di attesa) tra un'istruzione e l'altra e quelle 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

Strutture principali

I principali costrutti che utilizzeremo nei nostri sketch sono i seguenti:

Ciclo **for**: è un ciclo che esegue tutte le istruzioni contenute al suo interno un numero fissato di volte.

Ciclo **while**: è un ciclo che esegue tutte le istruzioni che sono al suo interno fino a quando è vera una determinata condizione logica.

Struttura **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.

RICORDA

Queste strutture vanno inserite sempre all'interno del void loop

Come scrivere un ciclo *for*

Abbiamo bisogno di:

- Variabile intera → indice del ciclo (nell'esempio sotto *i*)
- Variabile intera o numero che indica il numero di iterazioni del ciclo da compiere (nell'esempio sotto 100)
- step del ciclo (nell'esempio sotto lo step è pari a uno in quanto abbiamo inserito *i++*)

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

Tale ciclo fa lampeggiare 100 volte il led di Arduino collegato alla porta 13 a intervalli di 1s

Come scrivere un ciclo *for*

Se vogliamo incrementare lo step del ciclo di un numero diverso da 1 al posto di scrivere `i++` possiamo scrivere: `i = i + 2`

Esempio:

```
int N=100;  
int i;  
for(i=0;i<N;i=i+2)  
{  
digitalWrite(13, HIGH);  
delay(1000);  
digitalWrite(13,LOW);  
delay(1000);  
}
```

Proviamo i due esempi precedenti

Per provare i due esempi precedenti dobbiamo scrivere il void setup dove dichiariamo il pin 13 come output e lo poniamo allo stato di LOW e poi inseriamo gli sketch precedenti all'interno del void loop.

Se facciamo questo vediamo che il led non lampeggia un numero fissato di volte (N) ma lampeggia all'infinito.

Questo perché non appena il ciclo for termina, siccome siamo nel void loop, si ricomincia da capo.

RICORDA

Tutte le istruzioni vanno messe nel void loop.

Quindi la soluzione non è togliere il void loop oppure mettere il ciclo for al di fuori di esso!!

Come scrivere un ciclo *while*

```
while(condizione logica)
{
  istruzioni
}
```

Come scrivere un ciclo *while*

Esempio

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

Tale ciclo fa lampeggiare 100 volte il led di Arduino collegato alla porta 13 a intervalli di 1s

Come fermare il *void loop*

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.

In pratica l'abbiamo messo a dormire!

```
int b=1;  
while(b<2){}
```

Struttura *if*

if(condizione logica)

{

istruzioni

}

else if(condizione logica)

{

istruzioni

}

else

{

istruzioni

}

Se la condizione logica dell'*if* è soddisfatta vengono eseguite solo le istruzioni relative all'*if*. Se viene soddisfatta la condizione logica dell'*else if* vengono eseguite solo le relative istruzioni. Se nessuna delle due condizioni viene soddisfatta si eseguono le istruzioni nell'*else*.

Esempio di struttura *if*

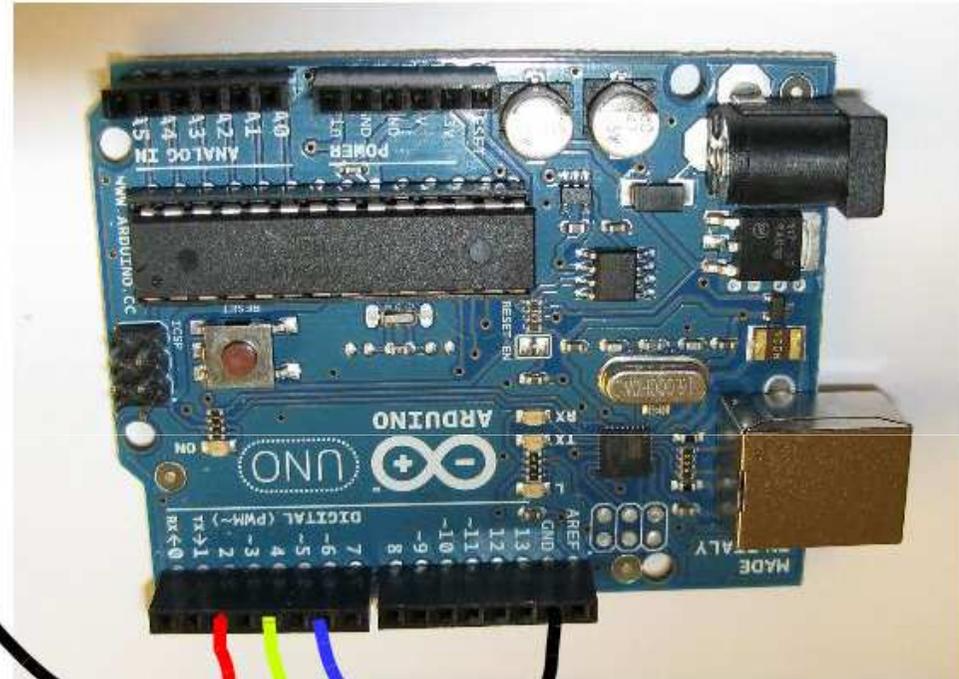
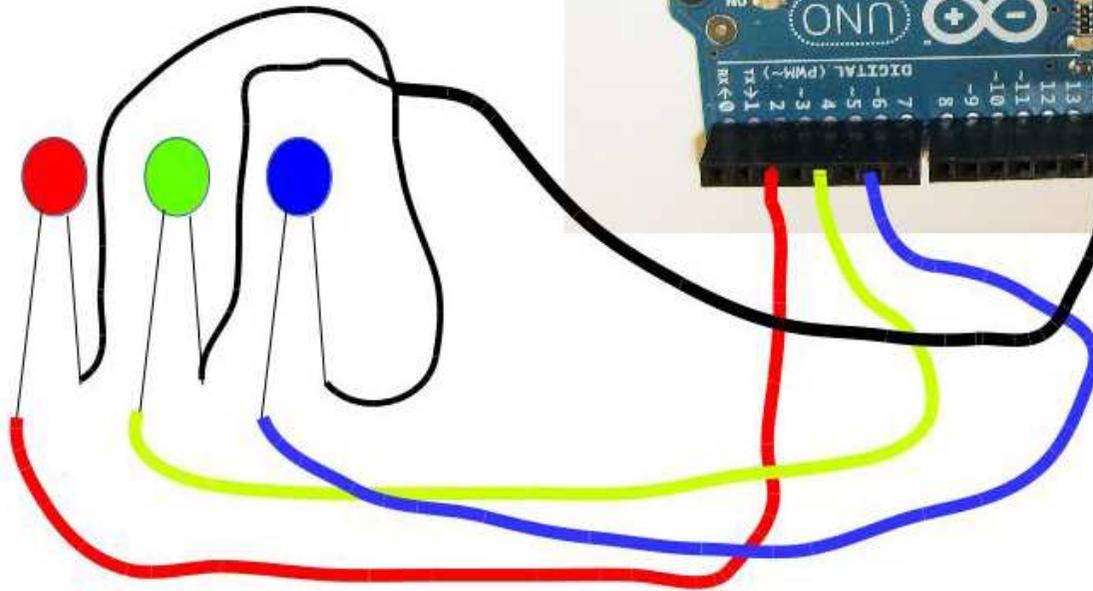
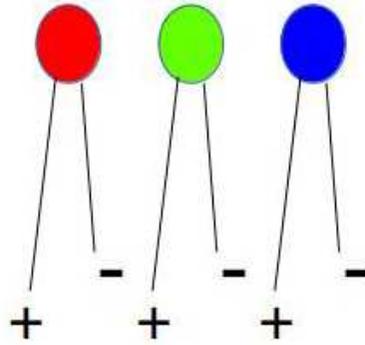
Supponiamo di avere tre led (rosso, verde e blu) e di collegarli rispettivamente alle porte digitali 2, 4, 6.

Un led ha due contatti ... quello positivo (quello più lungo) va collegato alla porta digitale mentre quello negativo (quello più corto) va collegato a massa (quindi alla porta GND di Arduino).

Supponiamo di avere inoltre un sensore i distanza (ad esempio il sensore a ultrasuoni che vedremo in seguito) che legge la distanza e la registra nella variabile *r*.

Esempio di struttura *if*

LED



Esempio di struttura *if*

Vogliamo accendere:

led rosso se $r < 30$ cm

led verde se $r > 50$ cm

led blu se $30 \text{ cm} < r < 50 \text{ cm}$

Esempio di struttura *if*

Questo è un tipico caso in cui è utile la struttura *if*.

```
void loop()
```

```
{
```

```
digitalWrite(2,LOW);
```

```
digitalWrite(4,LOW);
```

```
digitalWrite(6,LOW);
```

```
r = vedremo in seguito come misurarla
```

```
if(r<30) {
```

```
    digitalWrite(2, HIGH);
```

```
}
```

```
else if(r>50) {
```

```
    digitalWrite(4, HIGH);
```

```
}
```

```
else {
```

```
    digitalWrite(6, HIGH);
```

```
}
```

```
}
```

Operatori di confronto

Spesso nelle strutture *if* e *while* si ha la necessità di fare dei confronti tra variabili e costanti. Per fare ciò si utilizzano gli operatori di confronto.

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

Operatori logici

Nelle strutture if a volte si mettono non una ma due o più condizioni e ad esempio si richiede di eseguire l'if se una delle due, tutte e due o nessuna delle due è soddisfatta. In questo caso abbiamo bisogno degli operatori logici.

&& AND

|| OR

!! NOT

Esempi con operatori di confronto e logici

Supponiamo di avere due variabili x , y che sono lette in qualche modo con la scheda (ad esempio con qualche sensore o con gli ADC).

```
if(x <= 1 && y == 3)
```

```
{
```

```
  istruzioni (eseguite solo se x minore uguale di 1 e y uguale 3)
```

```
}
```

```
if(x > 1 || y != 3)
```

```
{
```

```
  istruzioni(eseguite solo se x maggiore di 1 o y diverso 3)
```

```
}
```

Esempi con operatori di confronto e logici

Supponiamo di avere due variabili x , y che sono lette in qualche modo con la scheda (ad esempio con qualche sensore o con gli ADC).

```
if(x <= 1 && y == 3)
```

```
{
```

```
  istruzioni (eseguite solo se  $x$  minore uguale di 1 e  $y$  uguale 3)
```

```
}
```

```
if(x > 1 || y != 3)
```

```
{
```

```
  istruzioni(eseguite solo se  $x$  maggiore di 1 o  $y$  diverso 3)
```

```
}
```

```
if(! x <= 1)
```

```
{
```

```
  istruzioni(eseguite solo se  $x$  non è minore uguale di 1)
```

```
}
```

Esempi con operatori di confronto e logici

```
if(! x <= 1)
```

```
{
```

```
  istruzioni(eseguite solo se x non è minore uguale di 1)
```

```
}
```

```
if(x > 1 || y != 3 || z < 5)
```

```
{
```

```
  istruzioni(eseguite solo se o x è maggiore di 1 o y è diverso  
  da 3 o z è minore di 5)
```

```
}
```

Come stampare i dati

Supponiamo di avere due variabili x e y su cui registriamo dei dati (supponiamo i valori di tensione letti nelle porte analogiche 1 e 2).

Se vogliamo stampare queste due variabili su monitor seriale possiamo usare il seguente codice (da mettere sempre all'interno del void loop).

```
void loop()  
{  
x = analogRead(A1);  
y = analogRead(A2);  
Serial.print("Tensione 1: ");  
Serial.print(x);  
Serial.print(" ");  
Serial.print("Tensione 2: ");  
Serial.println(y);  
}
```

Come stampare i dati

L'output dello sketch precedente sul monitor seriale è il seguente.

Tensione 1: 500 Tensione 2: 400

Tensione 1: 100 Tensione 2: 700

Tensione 1: 400 Tensione 2: 10

.....

I valori 500, 400, ecc. sono ovviamente solo indicativi.

RICORDA

i dati che compaiono sul monitor seriale non vengono memorizzati, se si ha bisogno di conservarli è necessario, con copia e incolla, incollarli in un file di testo.

Come stampare i dati usando *Gobetwino*

Riscriviamo il codice precedente con una sintassi che permette la stampa dei dati tramite ***Gobetwino*** (lanciato con il file go.bat come detto precedentemente) su un file di testo.

```
void loop()  
{  
x = analogRead(1);  
y = analogRead(2);  
Serial.print("#S|LOGTEST|[");  
Serial.print("Tensione 1: ");  
Serial.print(x);  
Serial.print(" ");  
Serial.print("Tensione 2: ");  
Serial.print(y);  
Serial.println("]#");  
}
```

Come stampare i vettori

Supponiamo di avere definito due vettori $t[100]$ e $v[100]$ (vettori con 100 componenti indicizzate però da 0 a 99) il primo contiene dei tempi e il secondo le velocità in quell'istante.

Supponiamo inoltre di voler stampare le componenti dei vettori su un file di testo contenente due colonne (la prima con le componenti di t e la seconda con le componenti di v) e come separatore vogliamo mettere il punto e virgola.

Supponiamo inoltre di voler mettere un'intestazione in cui si indichi che cosa rappresentano le due colonne.

OUTPUT:

tempi;velocità

t1; v1

t2; v2

t3; v3

t4; v4

.....

Come stampare i vettori

Per fare ciò possiamo utilizzare il seguente codice:

```
void loop() {  
  int i;  
  Serial.print("#S|LOGTEST|[");  
  Serial.print("tempi;velocità");  
  Serial.println("]#");  
  for(i=0;i<100;i++)  
  {  
    Serial.print("#S|LOGTEST|[");  
    Serial.print(t[i]);  
    Serial.print(";"); (se si vuole cambiare il separatore basta sostituire il ;)  
    Serial.print(v[i]);  
    Serial.println("]#");  
  }  
}
```

I sensori

I sensori che possono essere collegati alla scheda Arduino sono tantissimi e variano a seconda dello scopo dell'esperimento. Sono riportate solo le tipologie che utilizzeremo

Sensori di posizione

misurano la distanza tra il sensore e un ostacolo.

Sensori di temperatura

misurano la temperatura di un corpo.

Oltre a questi ne esistono moltissimi altri come: sensori di corrente, di pressione, di umidità, sensori in grado di rilevare la presenza di un essere vivente nei paraggi ecc ecc

Sensore di posizione a ultrasuoni hc-sr04

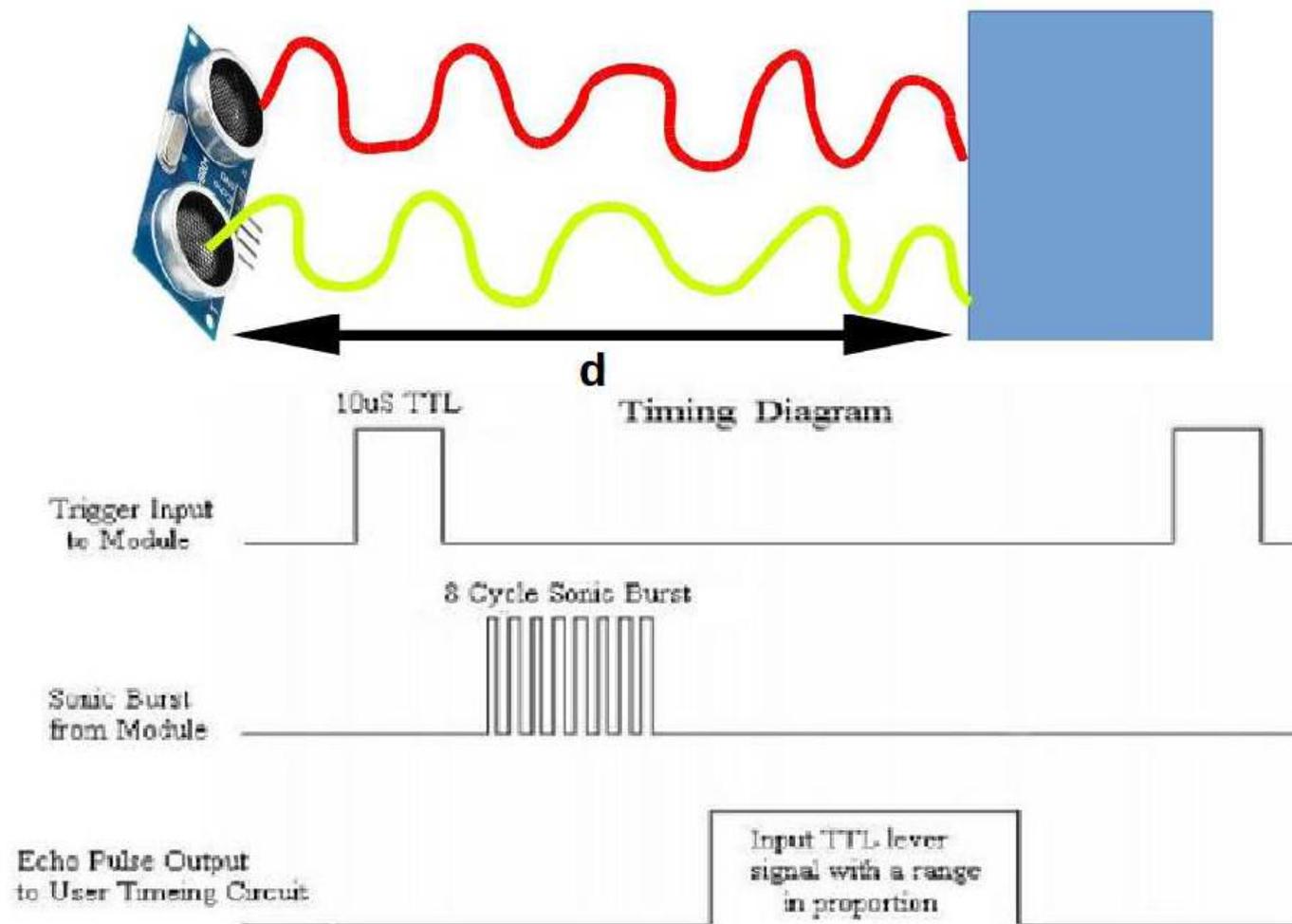


Sensore di posizione a ultrasuoni hc-sr04

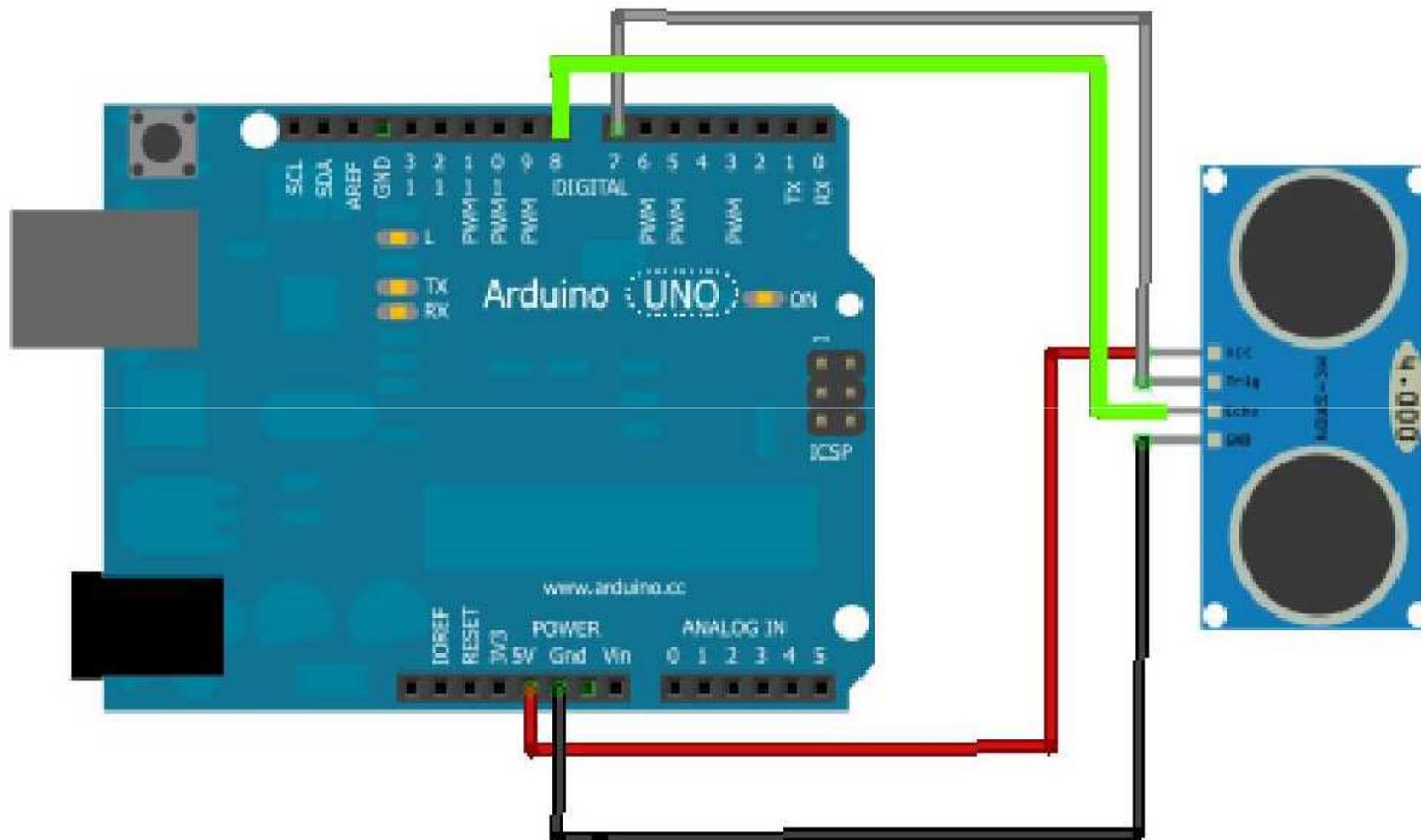
Il principio di funzionamento del sensore di posizione a ultrasuoni è basato sulla riflessione del segnale a ultrasuoni, generato dal sensore stesso, sul primo ostacolo che tale segnale incontra lungo il suo cammino.

Si può misurare il tempo impiegato dal segnale per compiere il percorso di andata (sensore-ostacolo) e ritorno (ostacolo sensore) e considerando il fatto che la velocità media degli ultrasuoni in aria a temperatura ambiente è 340 m/s si può calcolare la distanza dell'ostacolo dal sensore

Sensore di posizione a ultrasuoni hc-sr04



hc-sr04 : interfacciamento con Arduino



hc-sr04 : funzionamento

Il funzionamento di questo sensore a ultrasuoni si basa sui seguenti step:

- Si invia sulla porta digitale di trigger un impulso della durata di $10\mu\text{s}$ (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.

hc-sr04 : funzionamento

Con la funzione `pulseIn` possiamo misurare l'intervallo di tempo t in cui la porta di echo è stata nello stato HIGH (t corrisponde al tempo impiegato dal segnale per colpire l'ostacolo e tornare indietro). Si ha quindi che:

$$d = v_{\text{suono}} \cdot \frac{\Delta t}{2} = 340 \left[\frac{\text{m}}{\text{s}} \right] \frac{\Delta t}{2} [\text{s}]$$

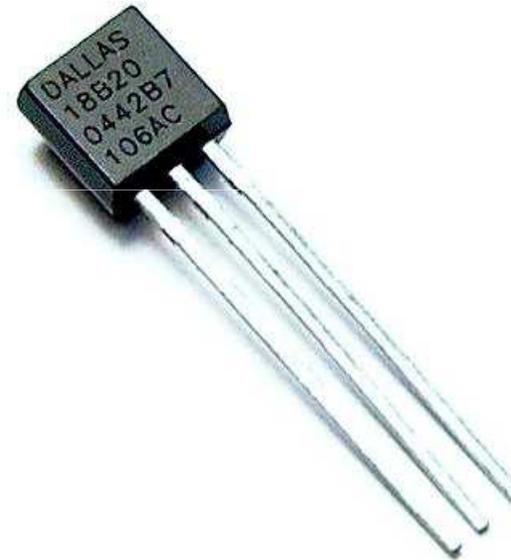
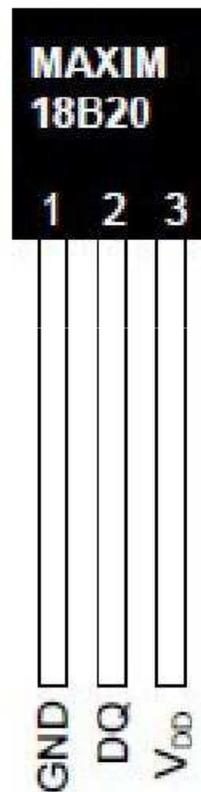
RICORDA: la funzione `pulseIn` ci dà l'intervallo di tempo t in microsecondi quindi è necessario convertirlo in secondi.

Un esempio di codice che permette la lettura della distanza utilizzando tale sensore è riportato nella cartella sketch all'interno della cartella PLS2016.

hc-sr04 : caratteristiche

- Costo: 4,39 € (Campustore)
- Max range: 4 m (massimo range effettivo: 2.5 m)
- Min range: 2 cm
- Errore: ± 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

Sensore di temperatura digitale: DS18B20



Sensore di temperatura digitale DS18B20

caratteristiche

Come tutti i termometri di questo tipo il principio di funzionamento è la variazione della resistenza del materiale di cui è fatto, al variare della temperatura.

- Range – 55°C +125°C
- Errore: $\pm 0.5^\circ\text{C}$
- Costo: 4.37 €

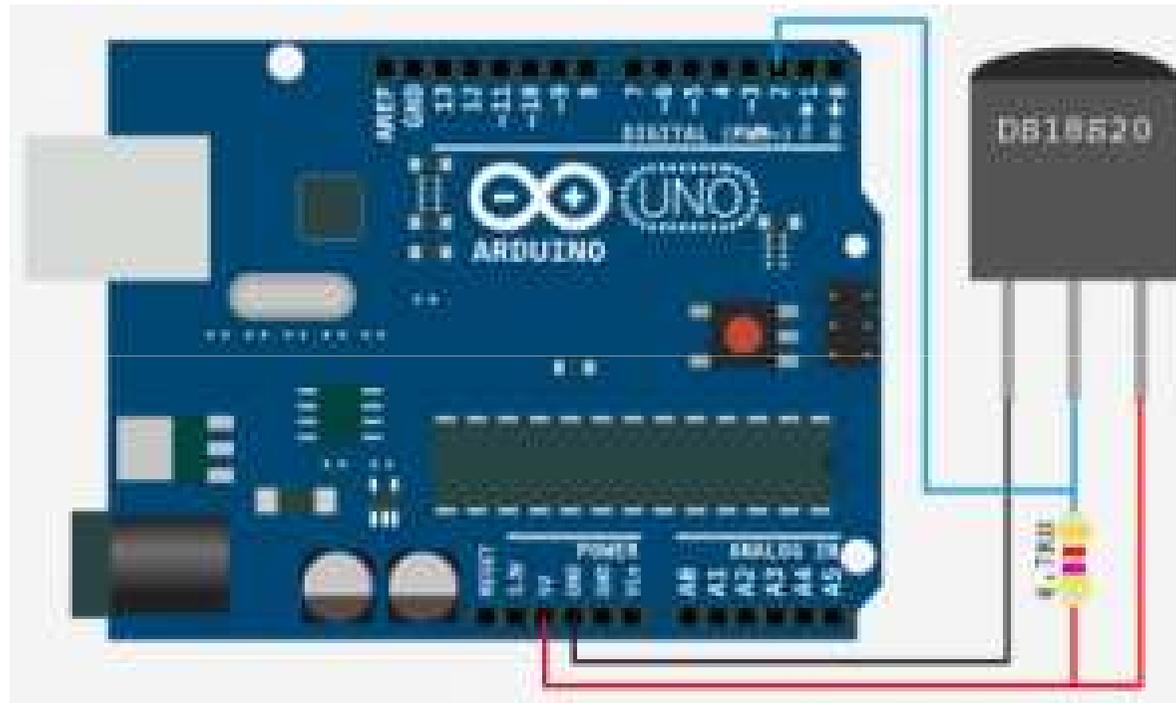
Si tratta di un termometro digitale che può essere usato con le seguenti librerie:

- DallasTemperature.h
- OneWire.h

Tali librerie verranno caricate precedentemente.

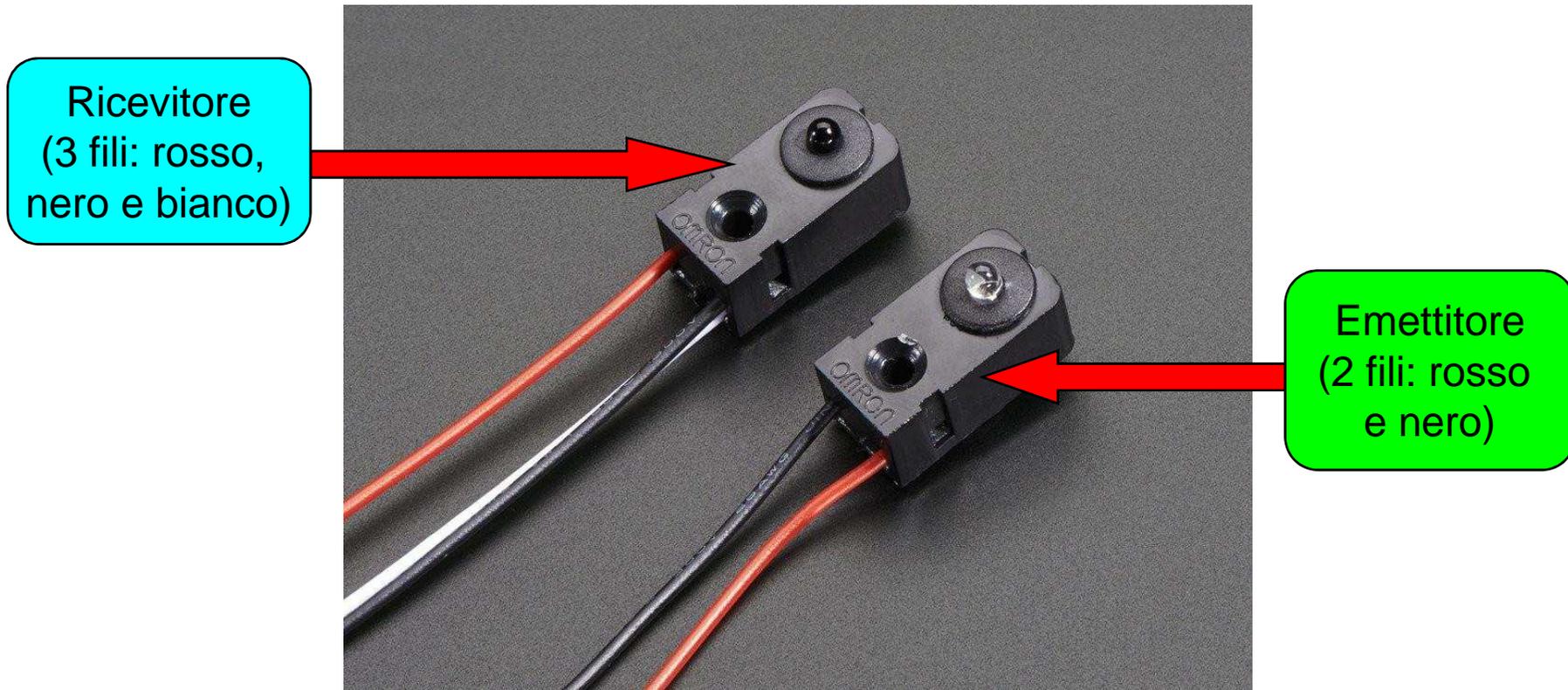
Comunque per importarle basta andare sull'Arduino ide menù sketch → importa libreria → importa libreria zip e poi selezionare la libreria da importare.

Sensore di temperatura digitale DS18B20 Interfacciamento con Arduino



Un esempio di codice per leggere la temperatura con questo tipo di sensore é contenuto nella cartella sketch all'interno della cartella PLS2016.

Sensore IR Break-beam



Ricevitore
(3 fili: rosso,
nero e bianco)

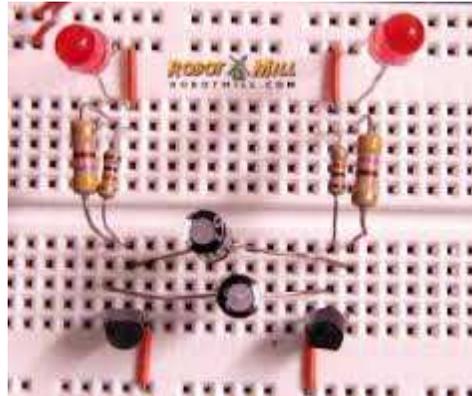
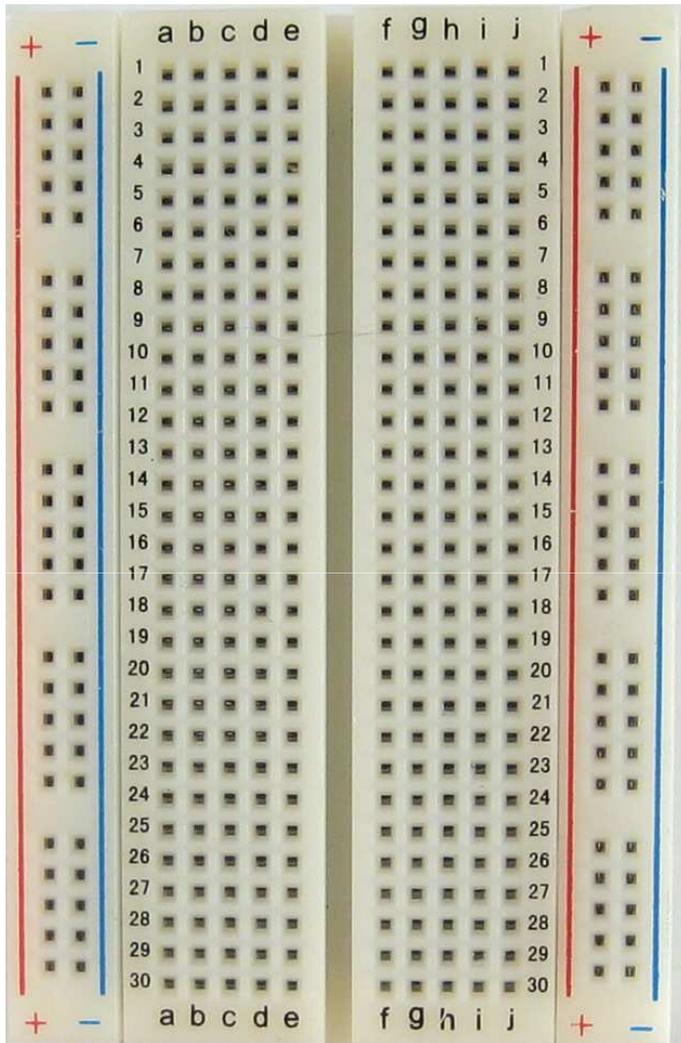
Emittitore
(2 fili: rosso
e nero)

I sensori a raggi infrarossi (IR) Break-beam sono costituiti da due elementi: un emettitore di un fascio IR e un ricevitore sensibile alla stessa radiazione. Quando qualcosa passa tra i due, e non è trasparente all'IR, allora il "fascio si rompe" (break-beam) e il ricevitore dà un segnale.

Useremo una versione IR da 5mm. Funziona fino a 25 cm di distanza.

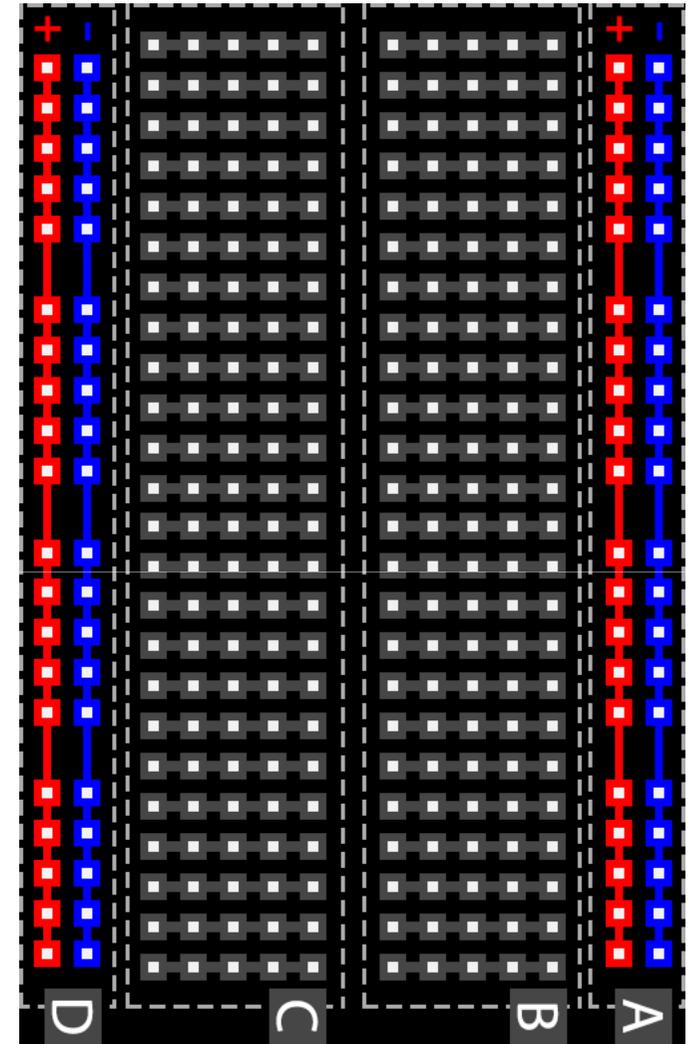
Si può alimentare da 3,3 V a 5 V.

La basetta *breadboard*



È il luogo fisico
dove costruiremo i
nostri circuiti.

Non serve fare
saldature.



I fori delle due file laterali sono collegati in verticale,
gli altri sono collegati, 5 a 5, in orizzontale.

FINE

GRAZIE PER L'ATTENZIONE!

<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>