Piano lauree scientifiche 2016

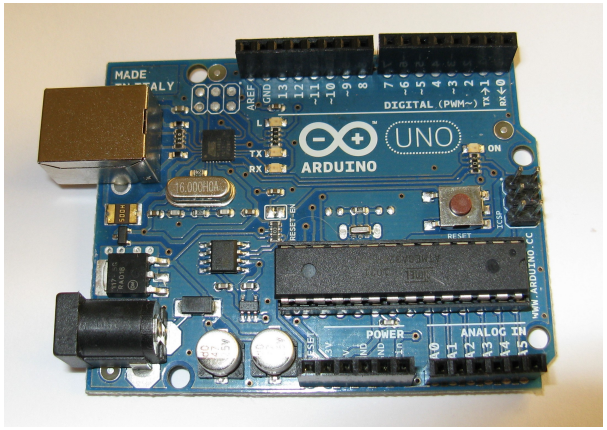
Fabio Capodaglio

20 Novembre 2015

Il laboratorio di fisica con Arduino:
introduzione ad Arduino e alla sua programmazione

Che cosa è Arduino?

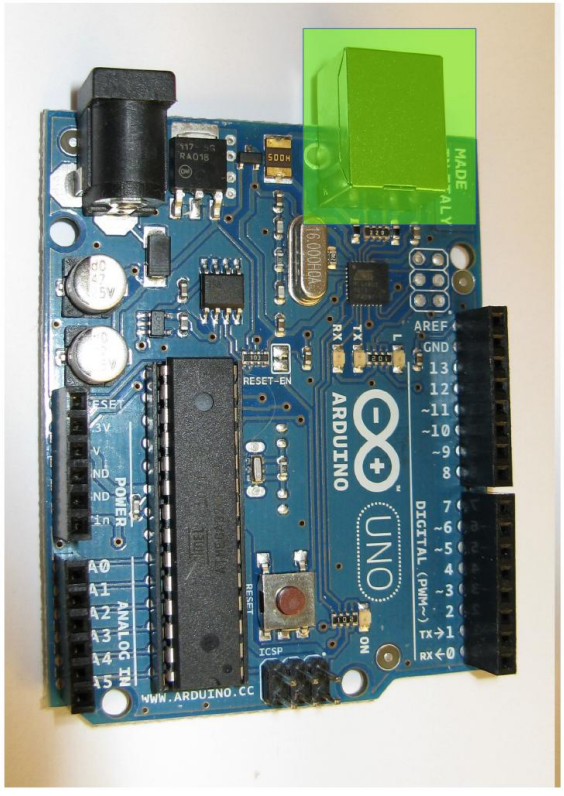
Arduino è una scheda elettronica di piccole dimensioni (comandata da un microprocessore) che presenta una serie di ingressi e uscite (pin) analogici o digitali che permettono di inviare e ricevere segnali.



La scheda nasce nel 2005 grazie al lavoro fatto di Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis. 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 32*Kb*

Alimentazione della scheda

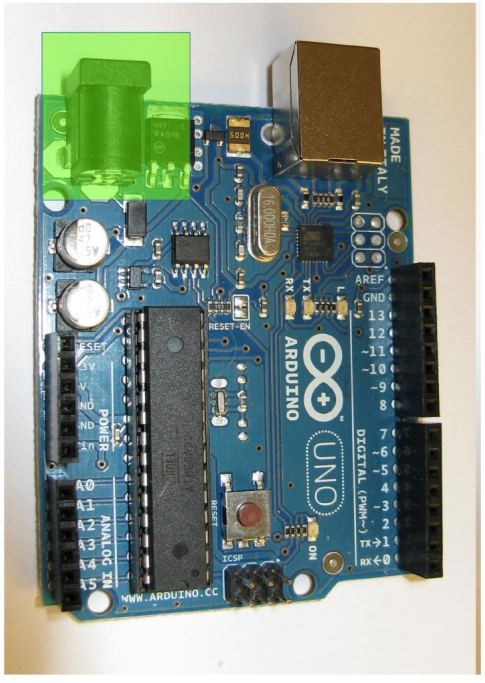
METODO 1: Collegando la scheda alla porta USB del computer



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

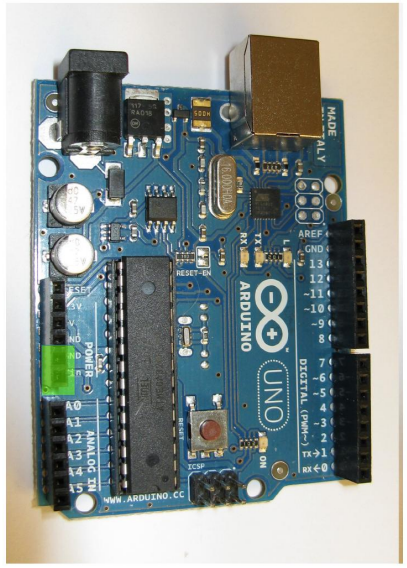
METODO 2: Collegando il connettore di alimentazione



Al connettore va collegata una tensione compresa tra i 7 V e i 12 V. Un regolatore ti tensione montato sulla scheda porterà tale tensione ai valori ideali (In questo caso la corrente può arrivare a 800 mA). Sul connettore è presente anche un diodo di protezione da eventuali inversioni della polarità.

Alimentazione della scheda

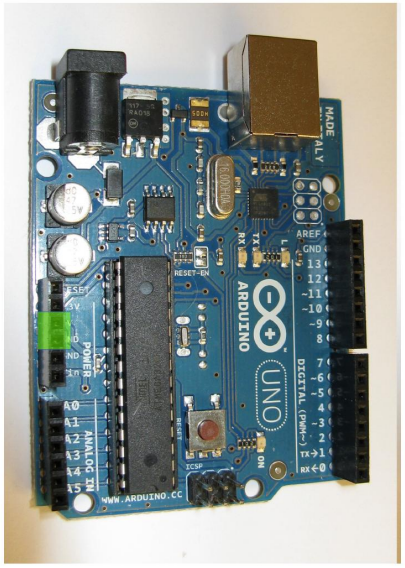
METODO 3: Attraverso i pin V_{in} e *GND*



In questo caso dobbiamo collegare una tensione compresa tra i 6.5 V e i 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 passa per il diodo di protezione quindi bisogna fare attenzione a non invertire le polarità per non bruciare la scheda!!!**

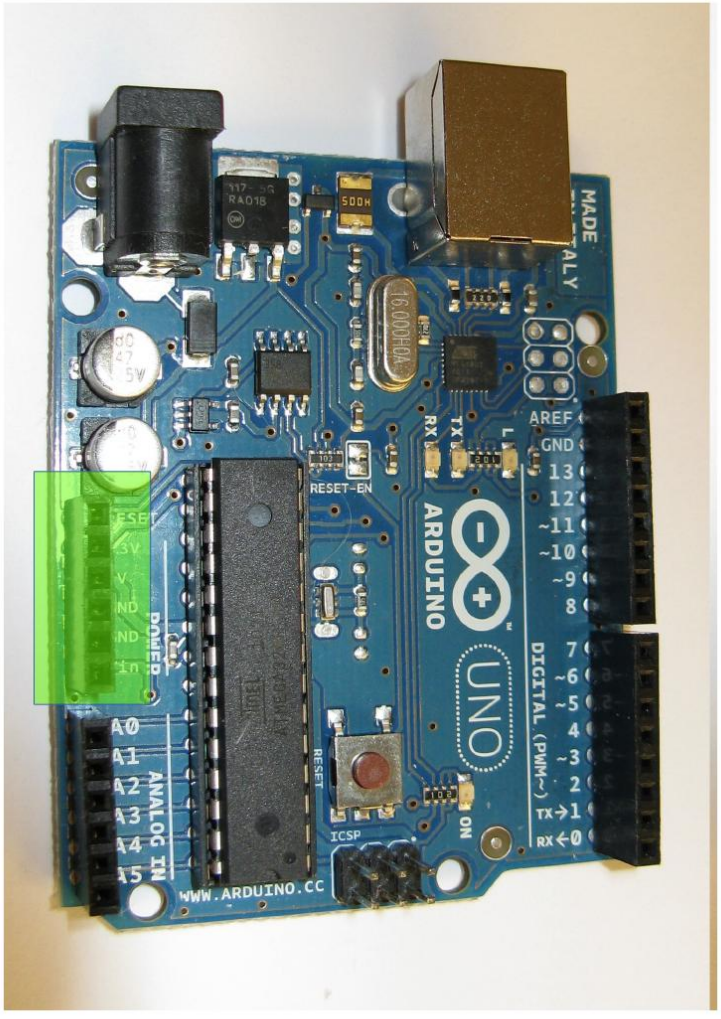
Alimentazione della scheda

METODO 4: Attraverso i pin 5 *V* **e** *GND*



In questo caso dobbiamo collegare una tensione compresa tra i 4.5 *V* e i 5.5 *V* (il polo positivo va collegato sul 5 *V* e il polo negativo sul *GND*). **In questo caso non si ha nè il regolatore di tensione ne il diodo di protezione quindi se si superano i** 5.5 *V* **oppure si invertono le polarità si rompe la scheda**.

Power pins


RESET
3V
V
GND
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
ATMEGA328
RESET
ICSP
ON
WWW.ARDUINO.CC
ARDUINO
UNO
DIGITAL (PWM~)
TX→1
RX←0
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
RESET-EN
RX
TX
L
16.000H-0A
500H
MADE

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*)

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* 3.3 *V* è un'uscita che fornice una tensione costante di 3.3 *V* e una corrente massima di 150 *mA*

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* 3.3 *V* è un'uscita che fornice 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)

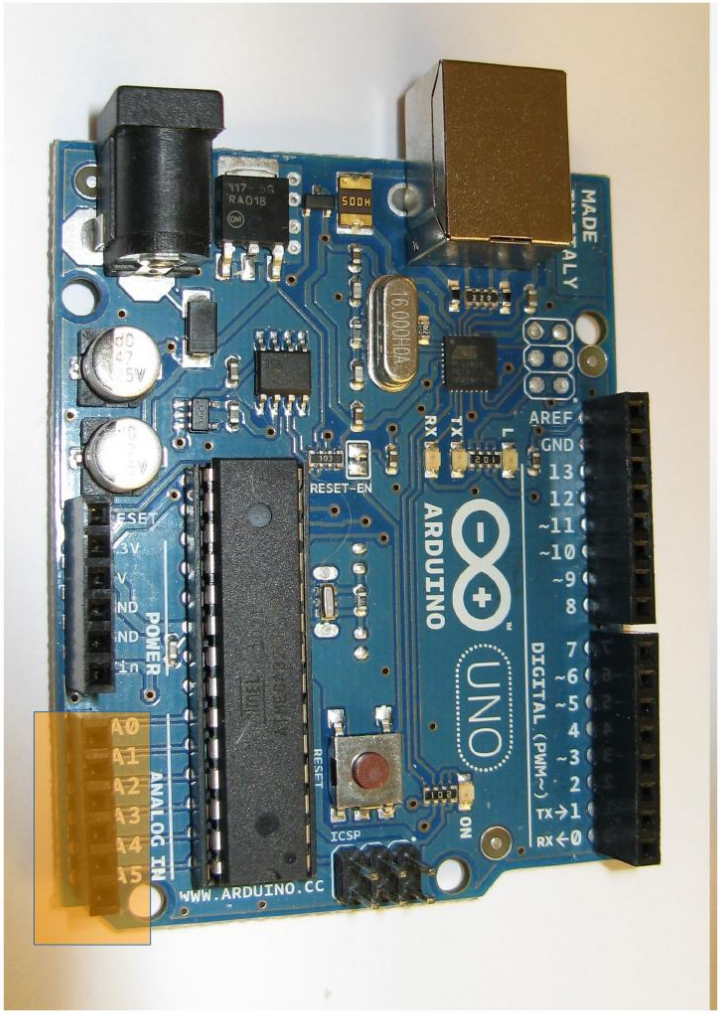
Power pins

- *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* 3.3 *V* è un'uscita che fornice 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

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* 3.3 *V* è un'uscita che fornice 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* V_{in}: si può usare per alimentare la scheda

Pin Analogici



Pin Analogici

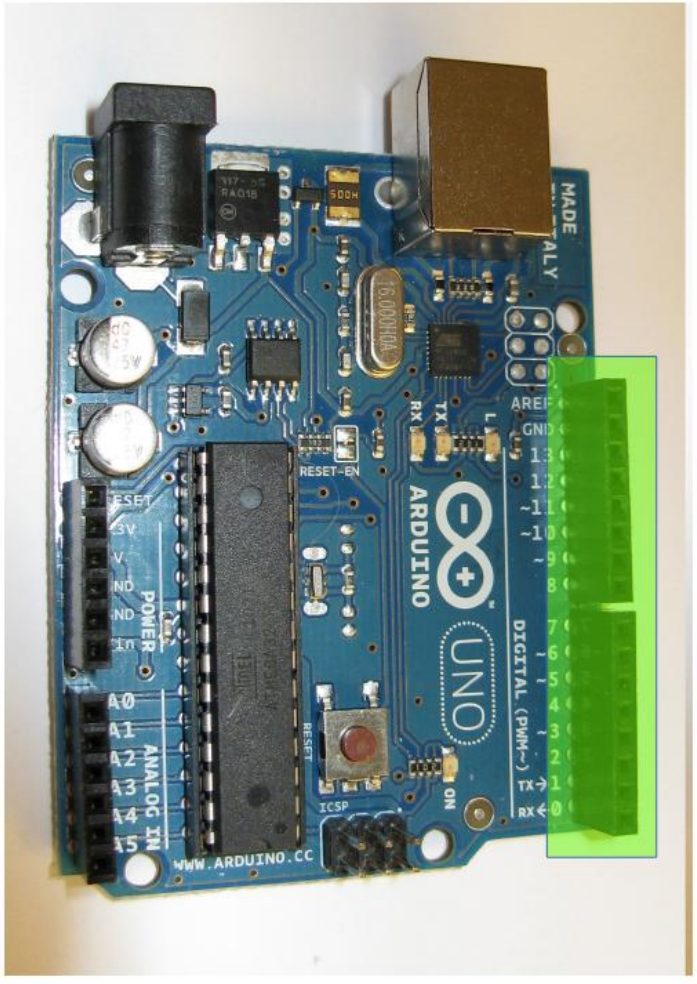
Arduino UNO REV3 presenta 6 pin analogici (tali pin sono di input).
Essi sono 6 convertitori analogico-digitale (ADC) con risoluzione di 10 bit. Ogni convertitore analogico digitale ha quindi un numero di canali pari a:

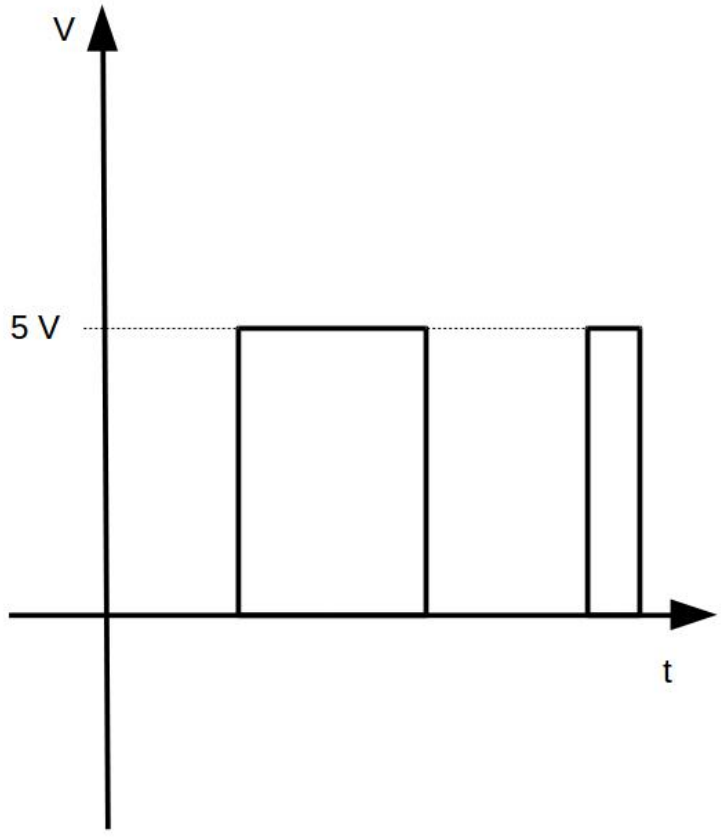
$$2^{10} = 1024$$

Gli ADC ci permettono di leggere un voltaggio nel range $0 - 5\ V$.
Tuttavia il comando di Arduino *analogRead* che ci 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 V possiamo usare la seguente relazione:

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

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 infatti 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*.

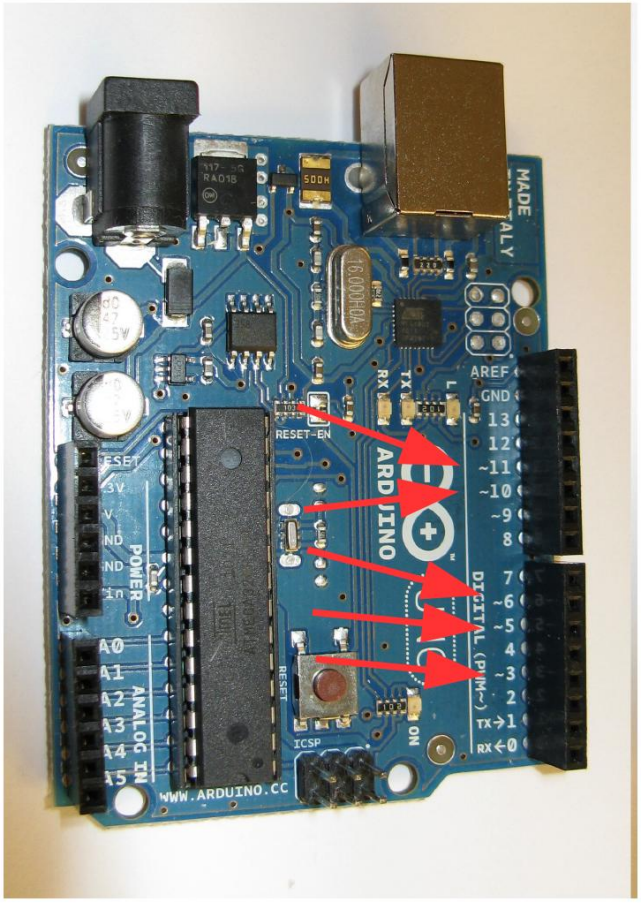
Pin digitali

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

- *LOW*: se riceve in ingresso un segnale di circa 0 *V*
- *HIGH*: se riceve in ingresso un segnale di circa 5 *V*

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

Pin digitali speciali → Pulse Width Modulation (PWM)



I pin digitali *PWM* sono quelli contrassegnati dal simbolo tilde (nell'*Arduino UNO REV3* sono i pin: 3, 5, 6, 10, 11)

Pin digitali speciali $\rightarrow$ Pulse Width Modulation (PWM)

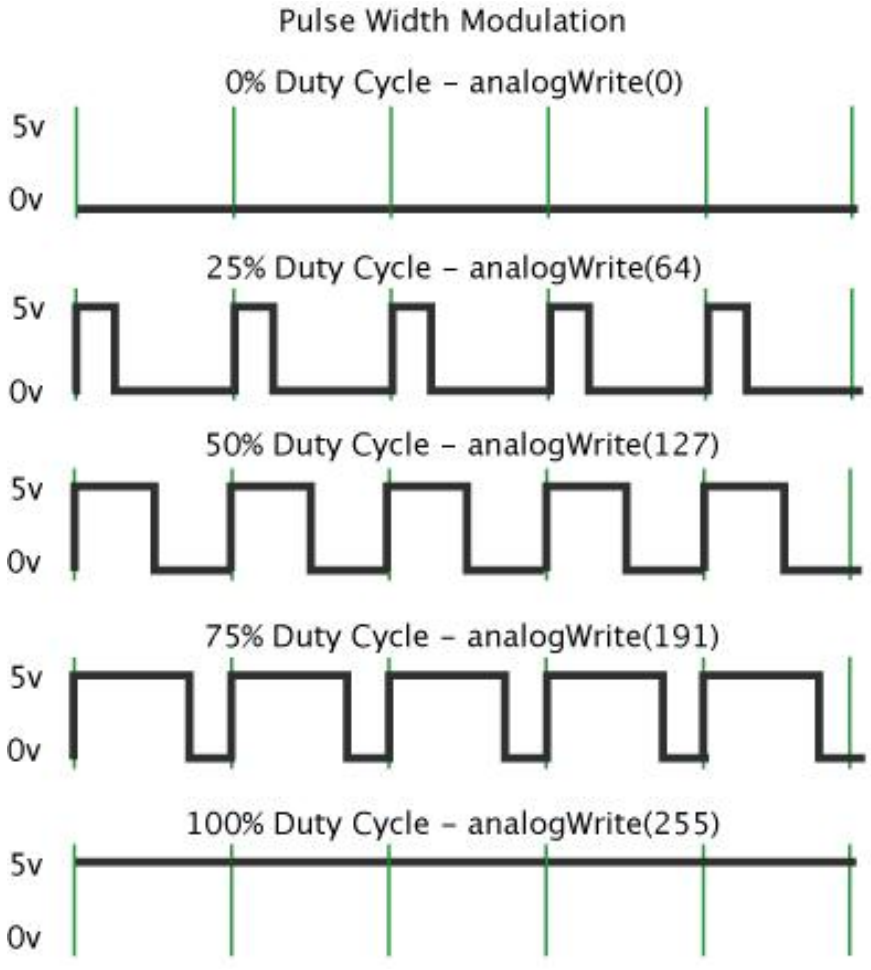
I pin *PWM* sono dei pin digitali che possono fornire in output una **tensione variabile** da 0 V a 5 V.
Analogamente al caso della lettura della tensione con le porte analogiche (per la lettura si usava *analogRead*) anche in questo caso la tensione va scritta in canali e non in Volt. Per farlo si utilizza 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:

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

Pin digitali speciali $\rightarrow$ Pulse Width Modulation (PWM)

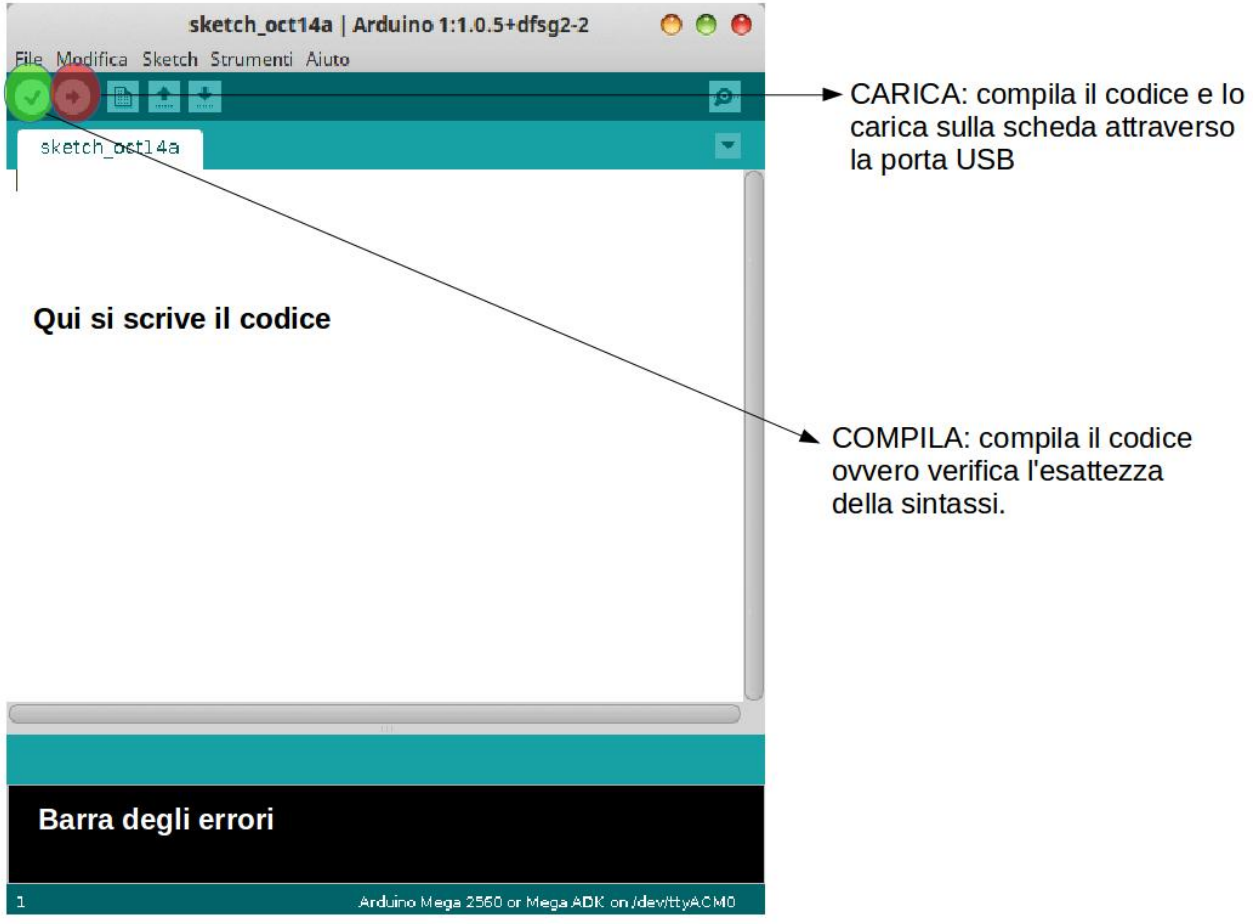
La tensione viene modulata attraverso il metodo della Pulse Width Modulation (modulazione di larghezza di impulso). Esso consiste in una serie di impulsi ad una determinata frequenza (e quindi a un determinato periodo T) che restano accesi per un tempo $\tau \leq T$. In base al valore di τ abbiamo quindi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

Pin digitali speciali $\rightarrow$ Pulse Width Modulation (PWM)



Come programmare *Arduino*???

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*???

Cosa fare se la compilazione va a buon fine ma il caricamento sulla scheda fallisce?

Come programmare *Arduino*???

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* $\rightarrow$ *scheda* $\rightarrow$ *Arduino uno* e la porta seriale dal menù *strumenti* $\rightarrow$ *porta*.

Come programmare *Arduino*???

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* $\rightarrow$ *scheda* $\rightarrow$ *Arduino uno* e la porta seriale dal menù *strumenti* $\rightarrow$ *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 ad eseguirlo! Per far si che la misura, o più in generale lo sketch, caricato sulla scheda venga eseguito a partire da un determinato istante di tempo abbiamo almeno tre modi:

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 ad eseguirlo! Per far si che la misura, o più in generale lo sketch, caricato sulla scheda venga eseguito a partire da un determinato istante di tempo abbiamo almeno tre modi:

- Spegnere e riaccendere Arduino togliendo e rimettendo l'alimentazione alla scheda

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 ad eseguirlo! Per far si che la misura, o più in generale lo sketch, caricato sulla scheda venga eseguito a partire da un determinato istante di tempo abbiamo almeno tre modi:

- Spegnere e riaccendere Arduino togliendo e rimettendo l'alimentazione alla scheda
- Premere il tasto reset

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 ad eseguirlo! Per far si che la misura, o più in generale lo sketch, caricato sulla scheda venga eseguito a partire da un determinato istante di tempo 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 fare in modo che *Arduino* inizi una misura quando vogliamo noi?

Una volta che il programma è stato caricato sulla scheda *Arduino* inizia immediatamente ad eseguirlo! Per far si che la misura, o più in generale lo sketch, caricato sulla scheda venga eseguito a partire da un determinato istante di tempo 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

Noi utilizzeremo l'ultimo modo in quanto ci permette anche di stampare i dati su file.

Come usare Gobetwino?

Gobetwino è un software che non ha bisogno di istallazione e che si trova all'interno di una cartella chiamata *Gobet* (che si trova sul desktop). Tale programma una volta lanciato esegue il reset della scheda e fa ripartire lo *sketch* dall'inizio.

Come usare Gobetwino?

Gobetwino è un software che non ha bisogno di istallazione e che si trova all'interno di una cartella chiamata *Gobet* (che si trova sul desktop). Tale programma 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; nel nostro caso il comando *LOGTEST*. Tale comando permette di stampare i dati presi con Arduino su un file di testo che abbiamo chiamato *prova pls.txt* e che si trova all'interno della cartella *Gobet*.

Come usare Gobetwino?

Gobetwino è un software che non ha bisogno di istallazione e che si trova all'interno di una cartella chiamata *Gobet* (che si trova sul desktop). Tale programma 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; nel nostro caso il comando *LOGTEST*. Tale comando permette di stampare i dati presi con Arduino su un file di testo che abbiamo chiamato *prova pls.txt* e che si trova all'interno della cartella *Gobet*.

Come usare Gobetwino?

Poiché tale comando appende i file alla fine di tutto ciò che è scritto in *prova pls.txt* abbiamo creato il file *go.bat* che una volta eseguito copia il contenuto di *prova pls.txt* in un altro file di testo *prova pls "data e ora".txt* che viene creato sempre nella cartella *Gobet*; a questo punto il contenuto del file *prova pls.txt* viene cancellato e viene lanciato Gobetwino che salverà i nuovi dati proprio in tale file. In questo modo si ha che gli ultimi dati presi sono salvati nel file *prova pls.txt* mentre le vecchie misure sono salvate nei file *prova pls "data e ora".txt*.

ATTENZIONE: se quando lanciamo *Gobetwino* **o** *Go.bat* **abbiamo problemi con la porta seriale** (ovvero si ha un errore in cui *Gobetwino* dice che la porta seriale impostata non esiste) ; **dobbiamo cambiare la porta seriale dal menù**
settings $\rightarrow$ *serial port*

ATTENZIONE ATTENZIONE

Una volta terminata la misura la finestra di *Gobetwino* va assolutamente chiusa!!!
Se rimane aperta infatti avremo degli errori sia quando proveremo a caricare un nuovo sketch con l'*Arduino ide* sia quando lanceremo una nuova misura con *Gobetwino* stesso!!!

PER EVITARE DI PERDERE UN QUARTO D'ORA A CAPIRE PERCHÉ IL COMPUTER SEMBRA IMPAZZITO (QUANDO INVECE HA RAGIONE LUI) È BENE CHIUDERE LA FINESTRA DI GOBETWINO APPENA NON SERVE PIÙ QUINDI APPENA FINITA LA MISURA

Metodo alternativo per lanciare una misura e stampare i dati!

Un metodo alternativo all'utilizzo di *Gobetwino* potrebbe essere l'utilizzo del **monitor seriale** di *Arduino* (menù *strumenti* $\rightarrow$ *monitor seriale*). Infatti una volta aperto il monitor seriale *Arduino* inizia da capo l'esecuzione dello *sketch* e inoltre si ha anche la possibilità di stampare i dati su tale finestra. Tuttavia si hanno due grossi svantaggi:

- Non si ha la possibilità di esportare 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.
- Se i dati sono davvero molti non si ha la possibilità di vederli tutti.

Per questo motivo utilizzeremo Gobetwino.

Struttura di uno *sketch* per *Arduino*

Struttura di uno *sketch* per *Arduino*

Come abbiamo visto nella figura precedente uno *sketch* per *Arduino* si divide in tre parti:

Struttura di uno *sketch* per *Arduino*

Come abbiamo visto nella figura precedente 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*

Come abbiamo visto nella figura precedente 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).
- **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 si definisce 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*

Come abbiamo visto nella figura precedente 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).
- **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 si definisce 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.
- **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* $\rightarrow$ *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>
#include <OneWire.h>
```

Definizione e dichiarazione 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 numero del pin 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:

- **int** $\rightarrow$ intero a 16 bit;
- **long** $\rightarrow$ intero a 32 bit;
- **float** $\rightarrow$ numero in virgola mobile a 32 bit;
- **double** $\rightarrow$ 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. Facciamo ora un esempio:

int pippo;
int pippo1 = 0:
double misura;

In pratica la struttura è la seguente:

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

ATTENZIONE ai punti e virgola: come nel linguaggio C i ; vanno messi 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 $\rightarrow$ 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*.

Cosa mettere nel *void setup*?

I comandi da mettere necessariamente nel *void setup* sono:

- **Serial.begin(9600);** $\rightarrow$ 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);** $\rightarrow$ se si usano delle porte digitali. Tale comando imposta le porte digitali usate o in INPUT o in OUTPUT.
- **bitClear(ADCSRA,ADPS0);** $\rightarrow$ 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.

Cosa mettere nel *void setup*?

Nel caso in cui si utilizzano delle porte digitali è opportuno specificare anche lo stato iniziale (ovvero lo stato in cui si troverà la porta una volta usciti dal *void loop*) della porta attraverso il seguente comando:

- **digitalWrite(nomeporta / numeroporta, HIGH / LOW);**
  $\rightarrow$ 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.

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*

Le funzioni di *Arduino* che possono essere inserite nel *void loop* sono moltissime e in queste slides vengono riportate solo le principali e quelle che utilizzeremo poi negli esperimenti. Nel caso in cui si ha bisogno di una funzione particolare oppure non si ricorda la sintassi o il funzionamento di una dato comando basta scrivere tale comando su **google** e si troverà senza problemi tutta la documentazione (anche ufficiale) su tale comando.

Comandi di *Arduino* per il *void loop* $\rightarrow$ Porte digitali

- **digitalWrite(nomeporta / numeroporta, HIGH / LOW);** $\rightarrow$ Comando per scrivere su una porta digitale;
- **digitalRead(nomeporta / numeroporta);** $\rightarrow$ Comando per leggere lo stato di una porta digitale;
- **pulseIn(nomeporta / numeroporta, HIGH/LOW);** $\rightarrow$ Tale comando ci da il tempo (in μ**s**) durante il quale una porta digitale è rimasta rispettivamente nello stato HIGH o LOW (es: pulseIn(4,HIGH); $\rightarrow$ ci da in μs il tempo che la porta digitale 4 resta nello stato HIGH)
  Tale funzione è molto utile con il sensore a ultrasuoni che useremo

Comandi di *Arduino* per il *void loop* $\rightarrow$ Porte digitali PWM e porte analogiche

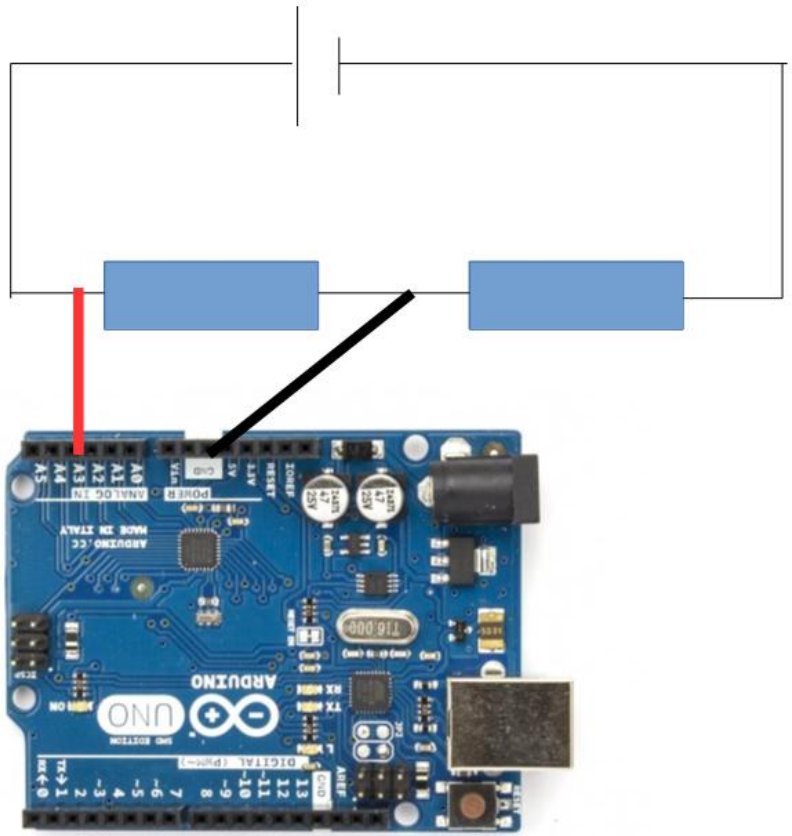
- **analogWrite(nomeporta / numeroporta, 0-255);** $\rightarrow$ Comando per scrivere su una porta digitale PMW (il valore 0 corrisponde a 0 V mentre il valore 255 corrisponde a 5 V);
- **analogRead(nomeporta / numeroporta);** $\rightarrow$ Comando per leggere il valore di tensione in ingresso a una porta analogica (il valore in uscita varia tra 0 $\rightarrow$ 0V e 1023 $\rightarrow$ 5V);

Come fare una misura di tensione con Arduino.

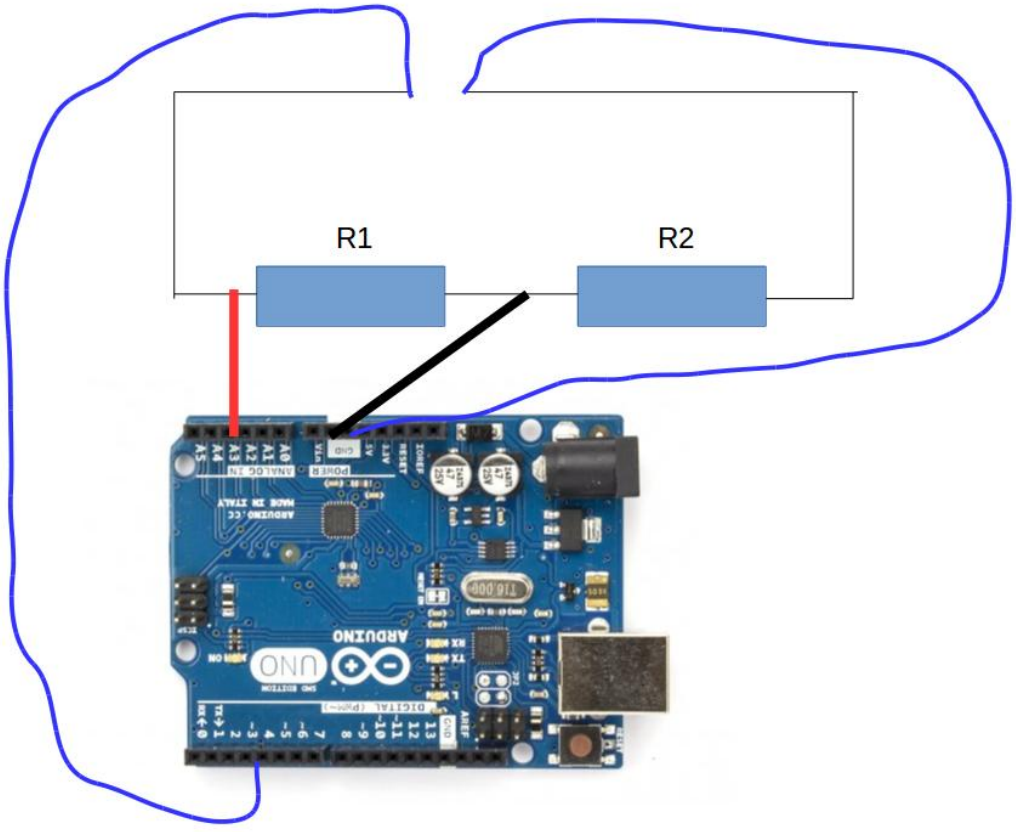
RICORDA: Quando si vuole fare una misura di tensione ai capi di un dispositivo (ad esempio una resistenza) si deve immaginare di usare *Arduino* come un multimetro; quindi i cavi da collegare sono due (il puntale rosso e quello nero del multimetro). Il puntale rosso è la nostra porta analogica mentre il puntale nero è una delle porte di ground (GND) di *Arduino*.

Attenzione a non invertire perchè *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

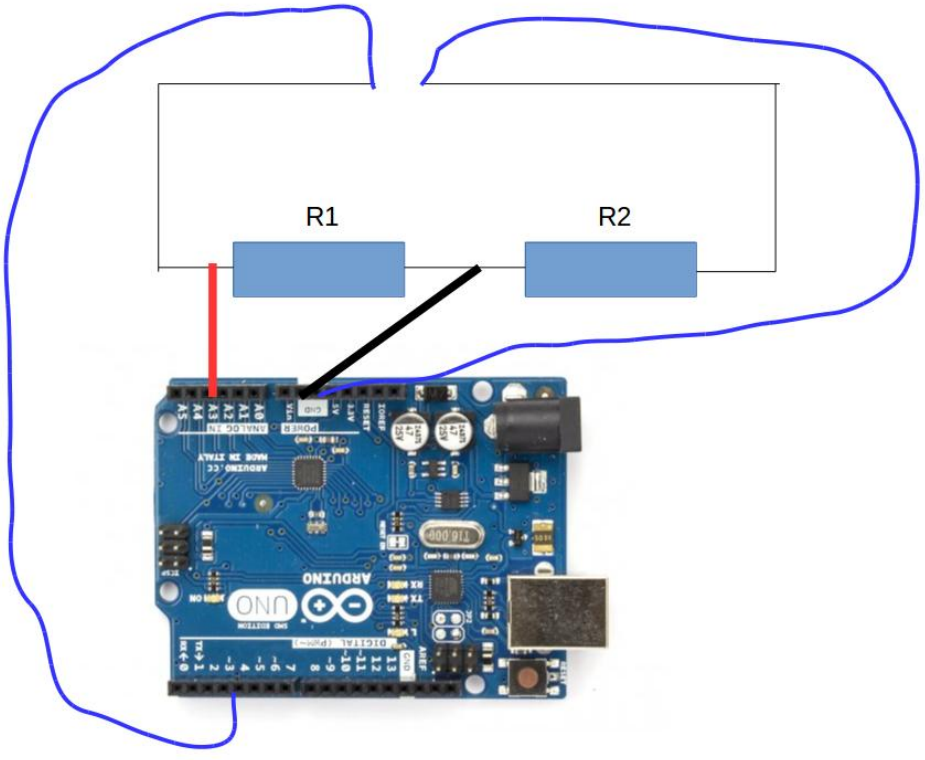
Come fare una misura di tensione con *Arduino*



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



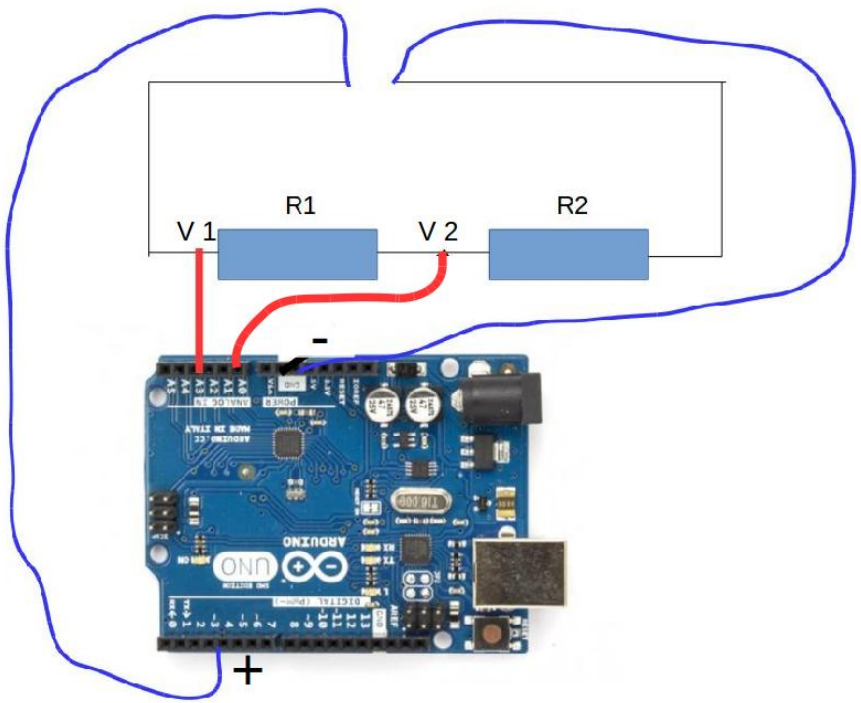
Il generatore di tensione potrebbe essere una porta digitale di *Arduino*!!!



ATTENZIONE: STO COMMETTENDO UN ERRORE!!!
COSÌ LEGGEREI 5V!!!

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

ECCO IL METODO CORRETTO!!!



La tensione ai capi si R_1 è: $V_1 - V_2$

Altre funzioni che incontreremo 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*.

Altre funzioni che incontreremo 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(variabile/numero)** $\rightarrow$ aspetta un numero di **millisecondi** pari al valore della variabile o del numero messo fra parentesi tonde (**che deve essere un intero**)

Altre funzioni che incontreremo 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(variabile/numero)** $\rightarrow$ aspetta un numero di **millisecondi** pari al valore della variabile o del numero messo fra parentesi tonde (**che deve essere un intero**)
- **delayMicroseconds(variabile/numero)** $\rightarrow$ aspetta un numero di **microsecondi** pari al valore della variabile o del numero messo fra parentesi tonde (**che deve essere un intero**)

Altre funzioni che incontreremo 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(variabile/numero)** $\rightarrow$ aspetta un numero di **millisecondi** pari al valore della variabile o del numero messo fra parentesi tonde (**che deve essere un intero**)
- **delayMicroseconds(variabile/numero)** $\rightarrow$ aspetta un numero di **microsecondi** pari al valore della variabile o del numero messo fra parentesi tonde (**che deve essere un intero**)
- **millis()** $\rightarrow$ restituisce in **millisecondi** il tempo trascorso da quando è stato lanciato lo *scketch*

Altre funzioni che incontreremo 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(variabile/numero)** $\rightarrow$ aspetta un numero di **millisecondi** pari al valore della variabile o del numero messo fra parentesi tonde (**che deve essere un intero**)
- **delayMicroseconds(variabile/numero)** $\rightarrow$ aspetta un numero di **microsecondi** pari al valore della variabile o del numero messo fra parentesi tonde (**che deve essere un intero**)
- **millis()** $\rightarrow$ restituisce in **millisecondi** il tempo trascorso da quando è stato lanciato lo *scketch*
- **micros()** $\rightarrow$ restituisce in **microsecondi** il tempo trascorso da quando è stato lanciato lo *scketch*

Strutture principali

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

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;

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;

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.

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: tali strutture vanno inserite sempre all'interno del *void loop*

Come scrivere un ciclo *for*?

Abbiamo bisogno di:

Come scrivere un ciclo *for*?

Abbiamo bisogno di:

- Variabile intera $\rightarrow$ indice del ciclo (nell'esempio sotto i)

Come scrivere un ciclo *for*?

Abbiamo bisogno di:

- Variabile intera $\rightarrow$ 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)

Come scrivere un ciclo *for*?

Abbiamo bisogno di:

- Variabile intera $\rightarrow$ 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++)

Come scrivere un ciclo *for*?

Abbiamo bisogno di:

- Variabile intera $\rightarrow$ 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);
}
```

Come scrivere un ciclo *for*?

Abbiamo bisogno di:

- Variabile intera $\rightarrow$ 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);
}
```

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);
}
```

Tale ciclo fa lampeggiare 50 volte il led di Arduino collegato alla porta 13 a intervalli di 1s

Proviamo ora i due esempi precedenti!

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

Proviamo ora i due esempi precedenti!

Per provare i due esempi precendenti dobbiamo scrivere il *void setup* dove dichiariamo il **pin 13 come output** e lo poniamo allo stato di **LOW**. 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.

Proviamo ora i due esempi precedenti!

Per provare i due esempi precendenti dobbiamo scrivere il *void setup* dove dichiariamo il **pin 13 come output** e lo poniamo allo stato di **LOW**. 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 possiamo fermarlo???

Per fermare *Arduino* possiamo usare un trucchetto 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){}
```

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

Struttura *if*

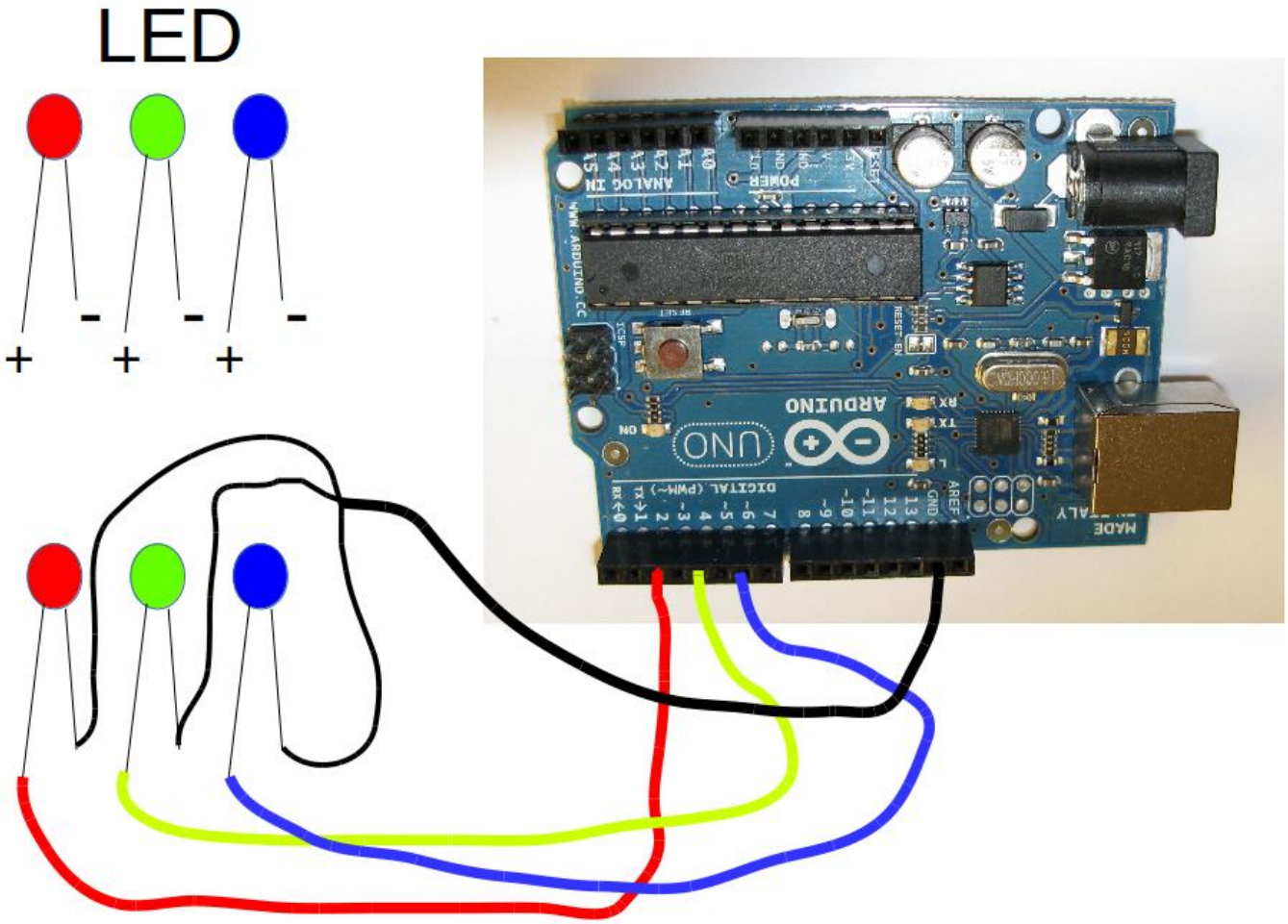
if(condizione logica**)**
{
istruzioni
}
else if(condizione logica**)**
{
istruzioni
}
else
{
istruzioni
}

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**. Ovviamente 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*



Esempio di struttura *if*

Vogliamo accendere:

- **led rosso** $\rightarrow$ **r** $<$ **30 cm**
- **led verde** $\rightarrow$ **r** $>$ **50 cm**
- **led blu** $\rightarrow$ **30 cm** $<$ **r** $<$ **50 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**.

- $==$ $\rightarrow$ **UGUALE**
- $!=$ $\rightarrow$ **DIVERSO**
- $>$ $\rightarrow$ **MAGGIORE**
- $>=$ $\rightarrow$ **MAGGIORE UGUALE**
- $<$ $\rightarrow$ **MINORE**
- $<=$ $\rightarrow$ **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**.

- && $\rightarrow$ **AND**
- || $\rightarrow$ **OR**
- ! $\rightarrow$ **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).

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)**
}

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

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** sarà il seguente.

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

Come stampare i dati usando *Gobetwino*

Ora riscriviamo il codice precedente con una sintassi che permette la stampa dei dati tramite *Gobetwino* (lanciato con il file *go.bat* come detto precedentemente).

```
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 dati usando *Gobetwino*

L'output dello *sketch* precedente sul file **prova pls** (contenuto nella cartella Gobet) sarà lo stesso di prima ovvero:

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

Come stampare i vettori usando *Gobetwino*

Supponiamo di avere definito due vettori **t[100] e v[100] (vettori con 100 componenti indicizzate però da 0 a 99)**. 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**). Come **separatore vogliamo mettere il punto e virgola**. Supponiamo inoltre di voler mettere un'intestazione in cui si indica cosa le due colonne rappresentano.

OUTPUT:

tempi velocità

t_1; v_1

t_2; v_2

t_3; v_3

t_4; v_4

.....

Come stampare i vettori usando *Gobetwino*

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 sostituirlo al ;)
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. Sotto sono riportati solo alcune di queste tipologie di sensori:

- **Sensori di posizione**: misurano la distanza tra il sensore e un ostacolo;
- **Fotocellule**: sono in grado di rilevare il passaggio di un oggetto attraverso di esse;
- **Sensori di temperatura**: misurano la temperatura di un corpo;

Oltre a questi ne esisto 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**

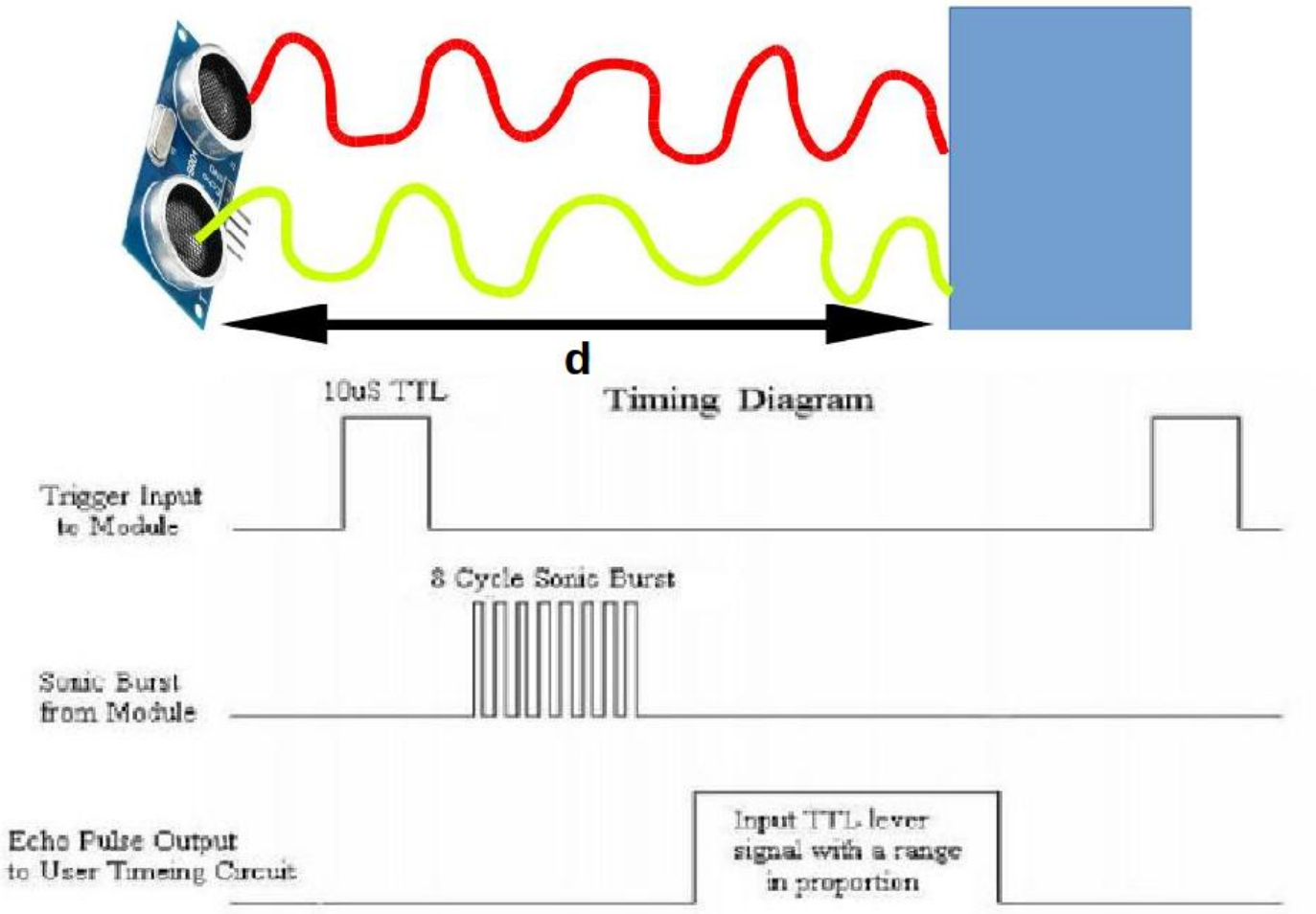
Sensore di posizione a ultrasuoni $\rightarrow$ hc-sr04

Sensore di posizione a ultrasuoni $\longrightarrow$ hc-sr04

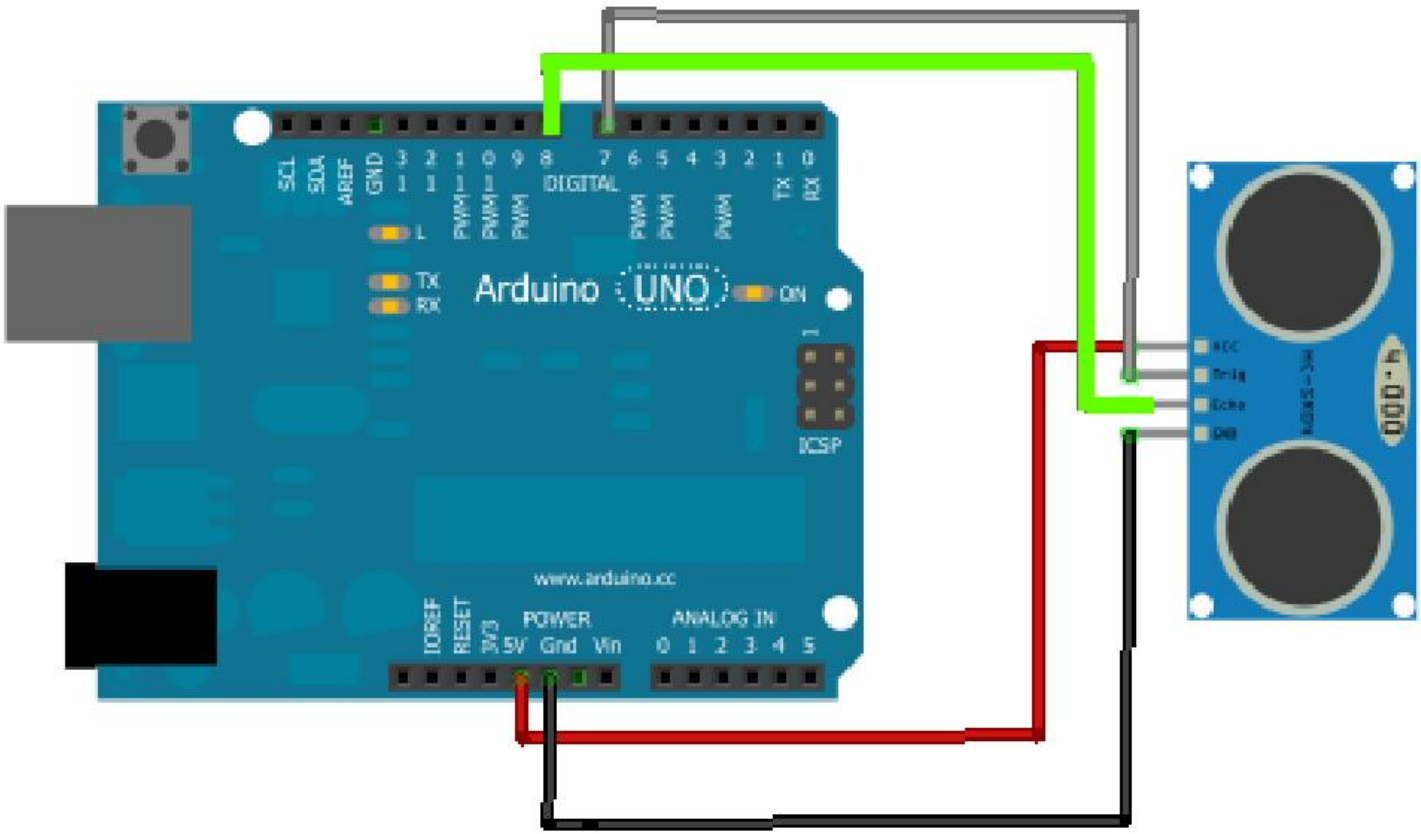
Il principio di funzionamento del sensore di posizione a ultrasuoni è basato sulla riflessione del segnale a ultrasuoni, generato dal sensore stesso, sul primo ostacolo che tale segnale incontra lungo il suo cammino. Conoscendo infatti il tempo impiegato dal segnale per compiere il percorso di andata (sensore-ostacolo) e ritorno (ostacolo sensore) e considerando il fatto che la velocità media degli ultrasuoni in aria è $340\ m/s$ **si può calcolare la distanza dell'ostacolo dal sensore**

In pratica è come tirare una pallina contro un muro e calcolare la distanza dal muro semplicemente dal tempo che la pallina impiega ad rimbalzare sul muro e tornare indietro (trascurando effetti di attrito).

Sensore di posizione a ultrasuoni → hc-sr04



hc-sr04 $\rightarrow$ interfacciamento con *Arduino*.



hc-sr04 $\rightarrow$ 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 μs (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 $\rightarrow$ funzionamento

Con la funzione *pulseIn* possiamo misurare l'intervallo di tempo Δt in cui la porta di echo è stata nello stato *HIGH* (Δt corrisponde al tempo impiegato dal segnale per colpire l'ostacolo e tornare indietro). Si ha quindi che:

$$d = v_s \cdot \frac{\Delta t}{2} = 340 \frac{m}{s} \cdot \frac{\Delta t[s]}{2}$$

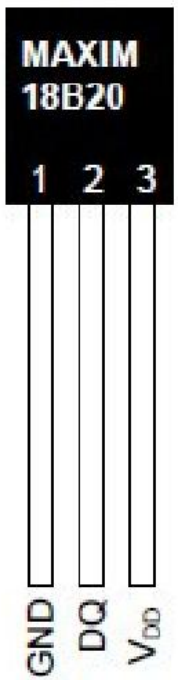
RICORDA: la funzione *pulseIn* ci dà l'intervallo di tempo Δt in μs quindi è necessario convertirlo in s.

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 $\rightarrow$ caratteristiche

- **Costo: 4,39 €** $\rightarrow$ Campustore
- Max range: 4 m $\rightarrow$ **Max range effettivo: 2.5 m**
- **Min range: 2 cm**
- **Errore:** $\pm$ **1 cm**
- **intervallo di tempo minimo tra una misura e l'altra: 25 ms**
- **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 il termometro al variare della temperatura.

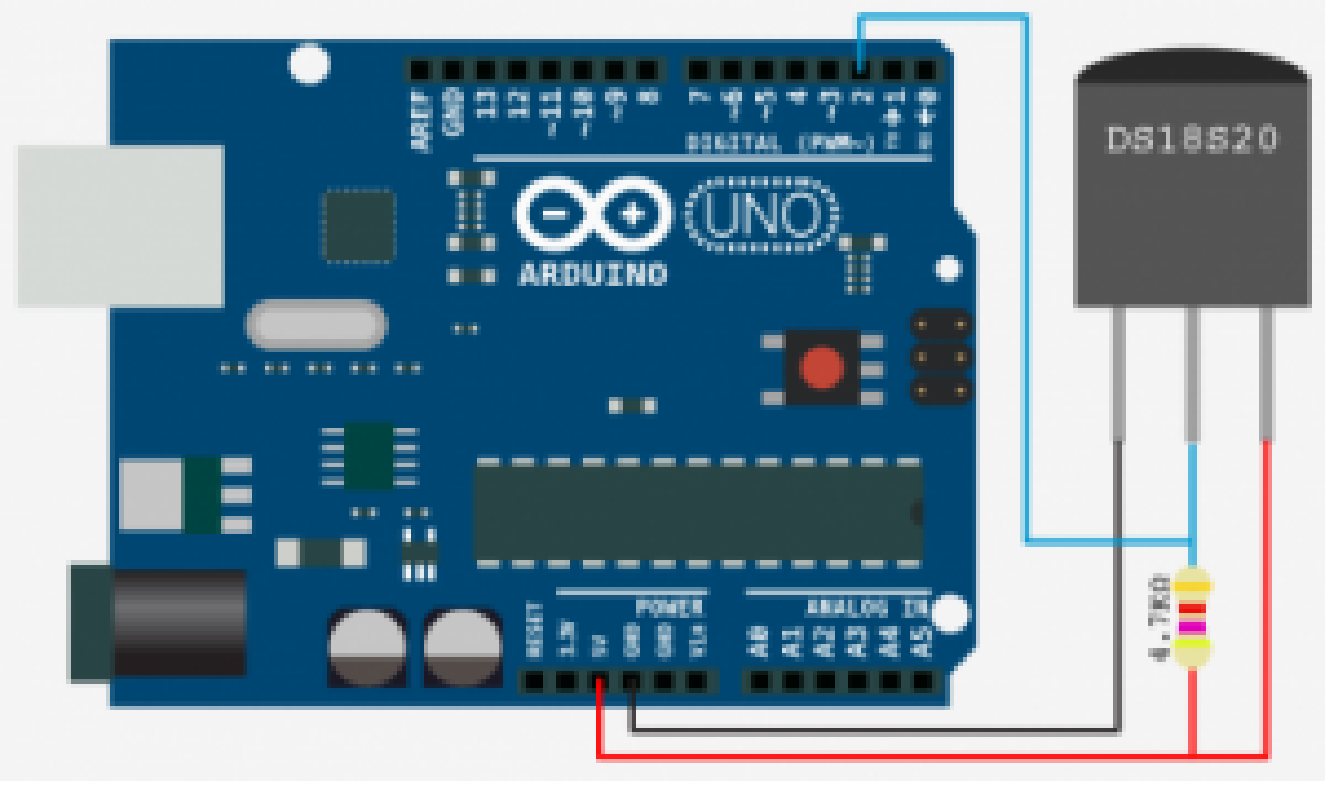
- **Range** $-55\,^oC\ -\ +125\,^oC$
- **Errore:** $\pm\,0.5\,^oC$
- **Costo: 4.37 €**

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

- DallasTemperature.h
- OneWire.h

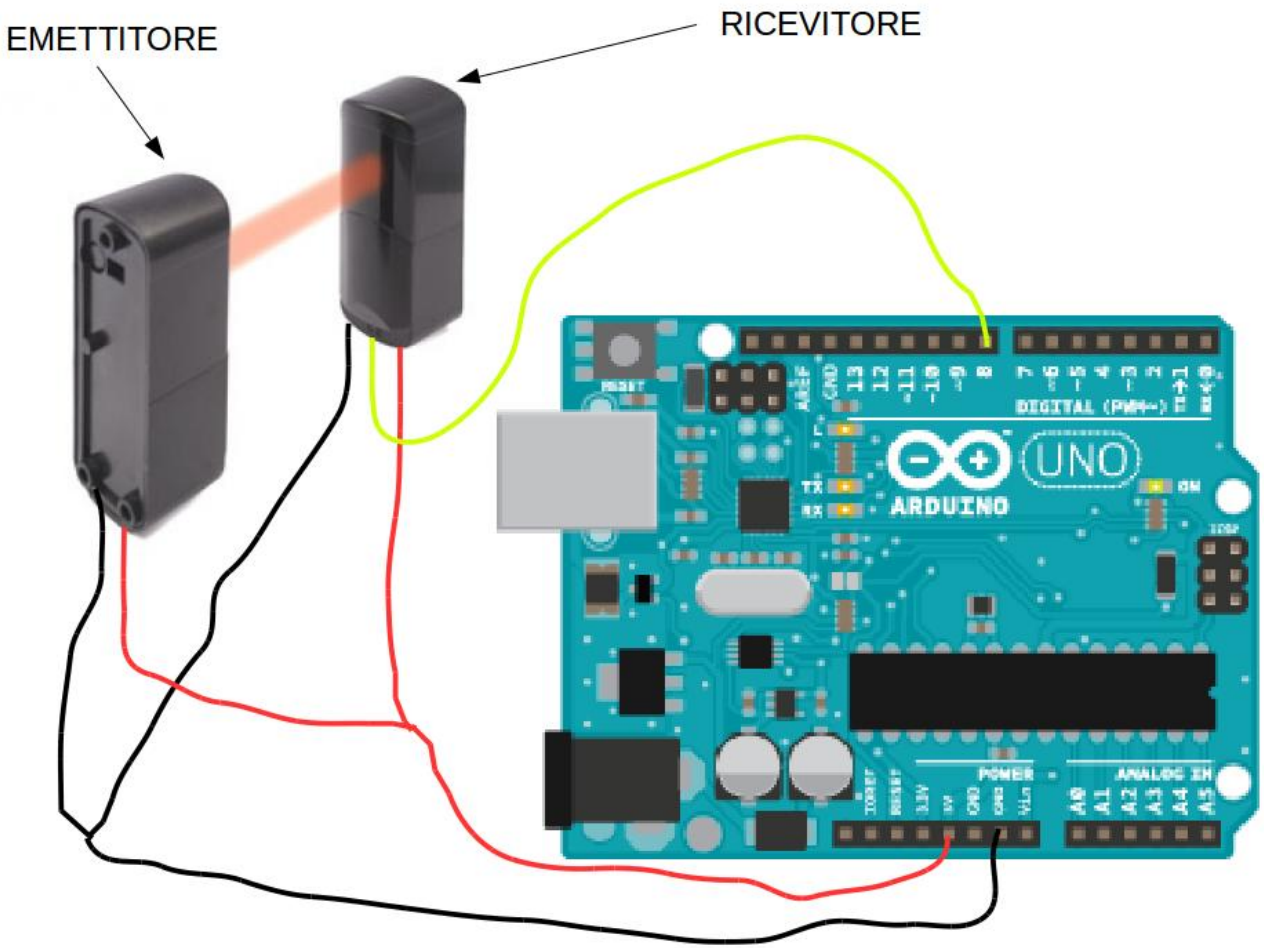
Tali librerie sono fornite nella cartella *librerie* (all'interno della cartella *PLS*2016) in formato *zip*. Per importarle basta andare sull'*Arduino ide* menù
sketch → *importa libreria* → *importa libreria zip* e poi selezionare la libreria da importare.

Sensore di temperatura digitale DS18B20 $\rightarrow$ interfacciamento con *Arduino*



Un esempio di codice per leggere la temperatura con questo tipo di sensore e contenuto nella cartella *sketch* all'interno della cartella *PLS*2016.

Fotocellule



GRAZIE PER L'ATTENZIONE!!!