

ELETTRONICA PRATICA

e

ARDUINO



PRIMO VOLUME

ELETTRONICA PRATICA e ARDUINO

di Maurizio Bazzano

Primo volume

**Titolo originale:
ELETTRONICA PRATICA e ARDUINO
foto copertina Maurizio Bazzano
© 2018 Elettronica pratica e Arduino
di Maurizio Bazzano
forzamao@gmail.com**

L'elettronica pratica è un metodo per imparare l'elettronica in modo semplice e diretto. Un manuale basato sulla semplicità discorsiva con molti esempi e formule semplificate pronte per essere usate. Senza le basi in elettronica è impossibile ottenere il massimo dal poliedrico mondo di Arduino.

Arduino è una scheda elettronica di prototipazione nata in Italia nel 2005 che permette agli sperimentatori di produrre innumerevoli realizzazioni.

In questo volume potrete apprendere la programmazione in C++ e i fondamenti della elettronica con molti esempi pratici.

Per poter fare tutte le esperienze descritte è necessario procurarsi il materiale elencato in ultima pagina.

Questo libro è stato scritto con Raspberry PI 3.

INDICE :

ELETTRONICA

| | |
|---|-----|
| Elettricitá | 9 |
| Corrente continua e alternata | 11 |
| Amper | 14 |
| Watt | 16 |
| Elettronica digitale | 19 |
| Le resistenze Unità di misura | 32 |
| Le resistenze A cosa servono | 35 |
| Le resistenze Serie e parallelo | 36 |
| Le resistenze Il potenziometro | 37 |
| Le resistenze Partitore di tensione | 39 |
| Le resistenze La fotoresistenza | 41 |
| Le resistenze La legge di Ohm | 42 |
| I diodi LED | 43 |
| La basetta di prova | 45 |
| I diodi | 141 |
| Il relé | 146 |
| Il transistor | 146 |
| Il MOSFET | 148 |
| Il foto accoppiatore | 155 |
| Il servomotore | 159 |
| I condensatori | 164 |

ARDUINO

| | |
|--------------------------------------|-----|
| Arduino | 17 |
| Collegare un LED | 46 |
| PWM | 62 |
| Collegare una fotoresistenza | 73 |
| Collegare un pulsante | 86 |
| Il display LCD | 110 |
| Bus I2C | 112 |
| Sensore di temperatura | 129 |
| Sensore Reed | 132 |
| Interrupt | 133 |
| Collegare un Relé | 149 |
| Collegare un transistor | 149 |
| Collegare un MOSFET | 153 |
| Collegare un foto accoppiatore | 157 |

INDICE :

PROGRAMMAZIONE

| | |
|---------------------------------------|-----------|
| Introduzione | 24 |
| Le uscite digitali | 27 |
| Le variabili | 50 |
| Convertitore analogico digitale | 57 |
| Gli ingressi analogici | 60 |
| Le uscite analogiche PWM | 65 |
| Il monitor seriale | 68 |
| Il controllo di flusso | 76 |
| Programmare un semaforo | 81 |
| Las entradas digitales | 88 |
| Scrivere un gioco | 91 |
| Le librerie | 114 |
| Programmare un ciclocomputer | 124 - 135 |
| PWM 2 | 158 |
| Servomotore | 169 |

ALTRO

| | |
|---------------------------------------|-----|
| Prologo | 7 |
| Listato completo gioco | 171 |
| Listato completo ciclo computer | 175 |
| Lista dei componenti del KIT | 177 |

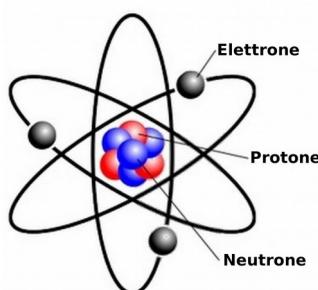
L'ELETTRICITÀ

Tutti i giorni usiamo la corrente elettrica e oggi sarebbe impensabile farne senza.

La usiamo in casa collegandoci alla presa di corrente, quando usiamo il telefono cellulare, in auto, la maggior parte delle cose che ci circondano sono elettriche.

In questo libro farò molti esempi usando l'acqua perché l'acqua è l'elemento più simile alla corrente elettrica.

Per capire cosa sia la corrente elettrica dobbiamo parlare dell'atomo perché è solo dall'atomo che possiamo ottenerla mettendo in movimento gli elettroni.



L'atomo è costituito da un nucleo di protoni con carica positiva e neutroni senza carica attorno ai quali orbitano gli elettroni che hanno carica negativa.

L'atomo ricorda un sistema planetario dove il nucleo è il sole e gli elettroni sono i pianeti che gravitano attorno.

Gli elettroni con carica negativa sono tenuti in orbita dai protoni positivi. A seconda dell'elemento a cui appartiene l'atomo possiede un numero definito di protoni, neutroni e elettroni.

Per esempio un atomo di idrogeno possiede solo un protone, un neutrone e un elettrone, l'atomo di rame ne possiede 29 e l'atomo dell'argento ne possiede 47.

Maggiore è il numero degli elettroni presenti nell'atomo e maggiore è il numero delle orbite attorno al nucleo.

Gli elettroni che orbitano vicino al nucleo sono chiamati "stabili" perché non si possono togliere facilmente dalla loro orbita, quelli lontani dal nucleo si chiamano "liberi" perché si possono spostare facilmente da un atomo ad un altro.

Questo spostamento di elettroni si può ottenerlo in modo meccanico (alternatori o dinamo) o chimicamente (pile e batterie)

Se a un atomo togliamo elettroni assume una carica positiva perché il numero dei protoni è maggiore degli elettroni. Se aggiungiamo elettroni ad un atomo la sua carica sarà negativa perché saranno in eccesso gli elettroni.

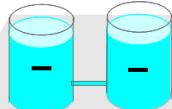
L'ELETTRICITÀ

Osservando una pila possiamo notare due terminali: uno Positivo “ + ” (maggior numero di protoni e uno negativo “ - ” (maggior numero di elettroni). Se colleghiamo un filo di rame ai terminali chiudiamo il circuito generando una corrente elettrica.

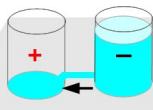
I protoni in eccesso del polo positivo attraggono gli elettroni in eccesso del polo negativo, quando tutti gli elettroni avranno rimesso in equilibrio gli atomi la corrente si fermerà e la pila si dirà che è scarica.

Per meglio intendere il movimento di questo flusso di elettroni possiamo usare due elementi come l'aria e l'acqua. Gli elettroni negativi li associamo all'acqua e i protoni positivi all'aria.

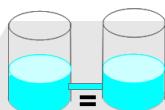
Se prendiamo due recipienti pieni di aria (polarità positiva) e li colleghiamo con un tubo alla base non avverrà nessun flusso perché in ambo i recipienti manca l'elemento opposto ovvero l'acqua.



Se colleghiamo due recipienti pieni di acqua (polarità negativa) non otterremo nessun flusso perché non esiste disequilibrio acqua / aria.



Se colleghiamo un recipiente pieno aria (polarità positiva) a uno pieno di acqua (polarità negativa) avremo un flusso di acqua dal recipiente pieno di acqua a quello pieno di aria.



Il flusso si fermerà quando i due recipienti avranno lo stesso livello di acqua e aria.

L'ELETTRICITÀ

La corrente elettrica è un flusso di elettroni attratti dai protoni. Quando tutti gli atomi avranno equilibrato i protoni con gli elettroni che mancano finirà la corrente elettrica. Ora è facile capire perché tutte le apparecchiature elettriche hanno due cavi di alimentazione di polarità opposta.



Tutte le pile hanno un terminale positivo e uno negativo perché nel loro corpo esiste un disequilibrio di elettroni. Questo disequilibrio si misura in Volt.

La tensione ha i suoi multipli e sottomultipli:

$$\text{Mega Volt (MV)} = \text{Volt} * 1,000,000$$

$$\text{Kilo Volt (KV)} = \text{Volt} * 1,000$$

$$\text{Mili Volt (mV)} = \text{Volt} / 1,000$$

$$\text{Micro Volt (} \mu\text{V) } = \text{Volt} / 1,000,000$$

La corrente elettrica è la circolazione di elettroni in un materiale conduttore che si muove sempre dal polo “-” al polo “+” della fonte di alimentazione. Però il senso convenzionale della corrente elettrica è al rovescio dal polo “+” verso il polo “-”. Questo criterio lo ereditiamo dalla storia, quando fu scoperta la corrente elettrica la comunità scientifica ignorava l'esistenza degli elettroni e dei protoni e decise un senso a caso sbagliando. In pratica questo errore non influisce minimamente nello studio e sviluppo di progetti.

In questo manuale i simboli matematici sono:

+ addizione

- sottrazione

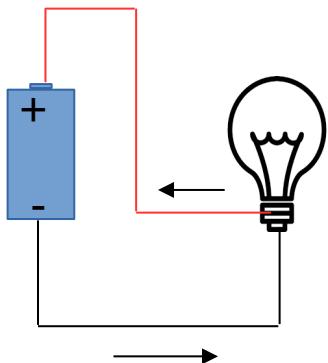
*** moltiplicazione**

/ divisione

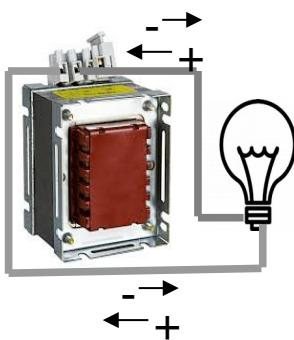
Sono i simboli che si usano per scrivere i programmi con l'IDE di Arduino.

CORRENTE CONTINUA E ALTERNATA

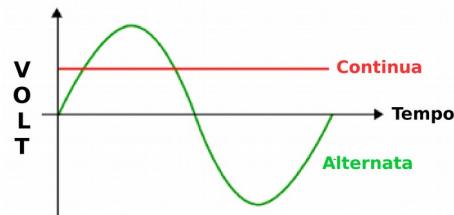
Tutti conosciamo il termine di corrente continua e corrente alternata però prima di spiegare la differenza vi ricordo che la corrente continua la preleviamo dalle pile, dalle batterie, dai pannelli solari e alimentatori specifici mentre la corrente alternata dagli alternatori, i trasformatori e dalla rete elettrica.



Se alimentiamo una lampadina con una corrente continua di una pila avremo due cavi uno con polarità negativa e l'altro con polarità positiva dove gli elettroni si muoveranno in una sola direzione.



Se alimentiamo una lampadina con una corrente alternata fornita da un trasformatore non avremo un cavo positivo e uno negativo perché la polarità cambierà continuamente. In entrambi i cavi correrà una corrente che da negativa passerà a positiva e viceversa. L'inversione di polarità è graduale.



La corrente continua (in rosso) è stabile nel tempo, la corrente alternata (in verde) inizia da un valore di zero Volt e gradualmente aumenta fino ad un massimo per poi abbassarsi fino allo zero, inverte la polarità e va al suo massimo negativo per poi tornare a zero e ricominciare.

CORRENTE CONTINUA E ALTERNATA

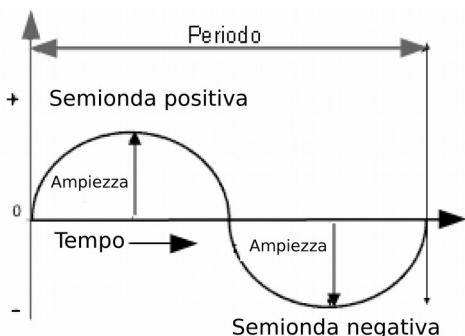
Nella metà alta della sinusoide gli elettroni scorrono in un senso, nella metà bassa in senso opposto. Questo movimento che si ripete nel tempo crea la corrente alternata. Una onda è composta da due semionde: una positiva e una negativa. Una semionda positiva e una semionda negativa formano una onda completa: un Periodo.

Il numero di onde che si ripetono nel tempo di un secondo si chiama Frequenza e si misura in Hertz (Hz).

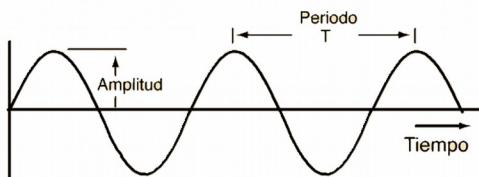
La corrente alternata di casa nostra è a 50 Hertz (Hz) quindi cambia di polarità 50 volte al secondo.

Le misure più utilizzate sono:

KHz (kilo Hertz) = 1,000 Hz,
MHz (Mega Hertz) = 1,000,000 Hz,
GHz (Giga Hertz) = 1,000,000,000 Hz.



Un'onda è composta da due semionde; una positiva e una negativa. Se in un secondo abbiamo una onda completa la sua frequenza sarà di 1 Hertz.



Se in un secondo abbiamo tre onde complete la frequenza sarà di tre Hertz.

CORRENTE CONTINUA E ALTERNATA



Solo con un oscilloscopio è possibile vedere e misurare un'onda.



Il frequenzimetro misura la frequenza di un'onda.

LA CORRENTE: unità di misura Amper

Il movimento degli elettroni dal negativo al positivo si chiama corrente e si misura in Amper (A). Per i più curiosi un Amper è uguale a $6,250,000,000,000,000,000$ elettroni che passano in un cavo in un secondo.

La corrente non dipende in alcun modo dalla tensione.

Per capire meglio la differenza tra tensione e corrente faccio un altro esempio con l'acqua:

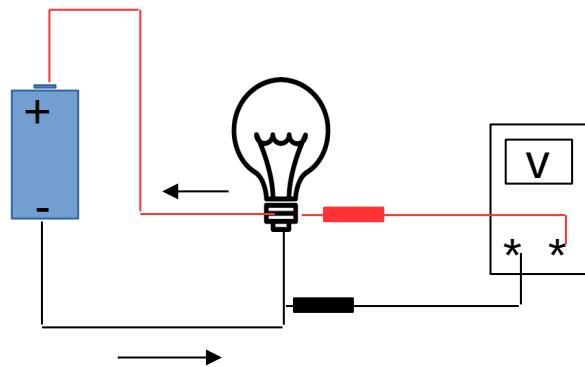
la tensione è la velocità dell'acqua che scorre in un tubo, la corrente è la quantità di acqua che passa nel tubo. Immaginiamo dell'acqua che passa in un tubo piccolo ad una certa velocità, in un secondo esce un litro di acqua. Ora immaginiamo un tubo più grande con acqua alla stessa velocità, da questo tubo in un secondo usciranno tre litri di acqua. La tensione è la velocità dell'acqua, la corrente è la quantità di acqua.

La corrente elettrica si misura in Amper (A) e i suoi sottomultipli.

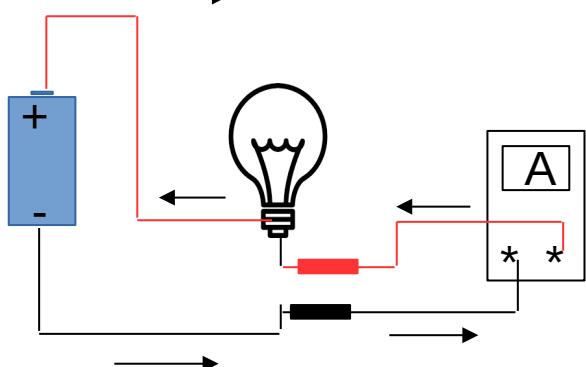
$$\text{milli Amper (mA)} = \text{Amper} / 1,000$$

$$\text{micro Amper (\mu A)} = \text{Amper} / 1,000,000$$

$$\text{nano Amper (nA)} = \text{Amper} / 1,000,000,000$$



Con un multmetro possiamo misurare la tensione in parallelo ai due poli.



Per misurare la corrente dobbiamo mettere in serie il multmetro interrompendo un polo.

LA POTENZA si misura in Watt

Conoscendo la tensione e la corrente assorbita da un utilizzatore possiamo conoscere il valore della potenza in Watt.
La formula che permette di calcolare i Watt é:

$$\text{Watt} = \text{Volt} * \text{Amper}$$

Un esempio: se abbiamo una lampadina da 12 Volt e 0.5 Amper in corrente continua consuma:

$$12 * 0.5 = 6 \text{ Watt}$$

Conoscendo i Watt e gli Amper possiamo conoscere la tensione:

$$\text{Watt} / \text{Amper} = \text{Volt}$$

$$6 / 0.5 = 12$$

conoscendo i Watt e i Volt possiamo trovare gli Amper;

$$\text{Watt} / \text{Volt} = \text{Amper}$$

$$6 / 12 = 0.5$$

Il multiplo dei Watt é:

$$\text{Kilo Watt (KW)} = \text{Watt} * 1,000$$

il sottomultiplo è:

$$\text{milli Watt (mW)} = \text{Watt} / 1,000$$

$$\text{micro Watt (} \mu\text{W} \text{)} = \text{Watt} / 1.000.000$$

Dobbiamo lavorare con attenzione con la corrente elettrica perché è come un fiume; un fiume può avere molta acqua (Amper) che si muove lentamente (Volt) e possiamo fare il bagno senza problemi però potrebbe avere poca acqua che passa rapidissima e non possiamo bagnarci perché ci trascinerà via uccidendoci.

La corrente elettrica non si vede e può uccidere!

La corrente elettrica non si vede e quando si sente è troppo tardi!

ARDUINO:

Ora che sappiamo cos'è la corrente elettrica possiamo parlare di Arduino.

Arduino è una scheda elettronica di prototipazione nata in Italia da una idea di Massimo Banzi che voleva una scheda elettronica semplice da usare e programmare per i suoi studenti.

Arduino ha una porta di comunicazione USB per collegarlo al computer che permette sia di alimentarlo che di caricare i programmi e ha una seconda connessione per una fonte di alimentazione esterna da 7 a 12V.

Ha 13 ingressi e uscite digitali e sei ingressi analogici, sei uscite analogiche PWM, una uscita di alimentazione a 5 e a 3.3V.

Ha porte di comunicazione UART, I2C e SPI.

ARDUINO:

Il cuore di Arduino Uno è un micro controllore prodotto da Atmel che si chiama Atmega328 e che ha un voltaggio operativo di 5V, una memoria **FLASH** da 32 Kilo Byte, dei quali 0.5 usati per il **Bootloader**, una **SRAM** da 2 Kilo BYTE e una memoria **EEPROM** da 1 Kilo BYTE.

La memoria FLASH è un tipo di memoria molto veloce che il micro controller usa per memorizzare il codice del programma e che non si cancella se manca alimentazione.

La memoria SRAM è una memoria dove Arduino mette i dati delle variabili e altri dati di calcolo che si cancella se manca la corrente.

La memoria EEPROM è una memoria che possiamo scrivere per conservare dei dati perché non si cancella se manca la corrente.

L'Atmaga328 è un micro controllore a 8 Bit.

Tutta l'elettronica digitale è binaria e ha solo due livelli di comunicazione ALT0 e BASSO quindi un Bit che è la parte più piccola di una informazione digitale può essere solo 1 o 0;

1 quando è ALT0, 0 quando è BASSO e non può avere altro stato.

1 significa 5 Volt, il + della tensione operativa.

0 significa 0 Volt, il - della tensione operativa

ATTENZIONE: 0 Volt non vuol dire che non è connesso a nulla deve essere collegato al negativo della tensione operativa del micro controllore.

Un gruppo di Bit si chiama BYTE.

L' ELETTRONICA DIGITALE:

Tutte le macchine digitali possono fare solo calcoli binari e per questo dobbiamo apprendere cosa sia la matematica binaria.

Arduino Uno, come tutti i micro controllori quando lo alimentiamo per la prima volta non capace di fare nulla. Tutti i micro controllori e i micro processori hanno bisogno di un programma, di un software che gli dica cosa fare.

Però per 'parlare' con un micro controllore occorre farlo nella sua lingua che è una serie di uno e zero che si chiama linguaggio macchina e non è pensabile di scriverlo.

Per questo esistono dei software che si chiamano IDE (Integrated Development Environment) che sono delle applicazioni che si incaricano di tradurre un linguaggio discorsivo umano in linguaggio macchina.

Per programmare Arduino usiamo l'IDE che è stato sviluppato in grado di comprendere il linguaggio di programmazione che si chiama C++.

Tutto il mondo delle macchine si basa solo su due numeri: l'uno e lo zero.

Noi usiamo una matematica di tipo decimale che significa che abbiamo 10 numeri per la prima cifra (prima decina) e per tutte le altre.

00 / 01 / 02 / 03 / 04 / 05 / 06 / 07 / 08 / 09

10 / 11 / 12 / 13 / 14 / 15 / 16 / 17 / 18 / 19

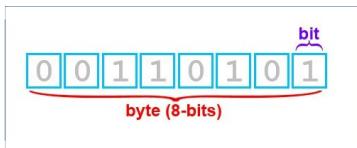
In matematica la prima cifra è sempre zero.

In matematica binaria i numeri possono aumentare solo di due unità.

0000 / 0001 / 0010 / 0011 / 0100 / 0101 / 0110 / 0111 / 1000 / 1001

0 1 2 3 4 5 6 7 8 9

L' ELETTRONICA DIGITALE:



La parte più piccola di questo numero si chiama **bit**, un gruppo di **bit** formano un **Byte**.

In elettronica digitale si possono trovare Byte da 4, 8, 16, 32 o 64 bit. Arduino Uno ha una architettura a 8 bit.

Capiere come funziona la matematica binaria è molto semplice perché ogni numero ha un suo peso.

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |

bit

Byte

Per conoscere il numero decimale del Byte in figura facciamo la somma dei pesi che hanno un 1 scritto sotto. Come i numeri decimali la cifra più piccola si trova a destra.

In figura abbiamo un 1 sotto i pesi 1, 4, 16 e 128. Se sommiamo i pesi otteniamo:

$$1+4+16+128=149$$

il Byte 10010101 è uguale al numero decimale 149.

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

bit

Byte

Se abbiamo un Byte di 8 bit con tutti i bit a 1 e sommiamo i pesi otteniamo:

$$1+2+4+8+16+32+64+128=255$$

Il numero più grande che si può scrivere con un Byte di 8 bit è 255.

Come avete avuto modo di vedere leggere un Byte è molto semplice ma se dobbiamo scriverlo?

Se per leggere un Byte abbiamo fatto la somma dei pesi dei bit, per scriverlo dobbiamo fare la sottrazione.

Immaginiamo di voler scrivere in binario il numero 117, per farlo dobbiamo partire da sinistra dal peso maggiore.

L' ELETTRONICA DIGITALE:

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Byte | | | | | | | |
| 0 | | | | | | | |

Il peso a sinistra è il più grande (128), dobbiamo sottrarre al nostro numero (117) il peso (128) però non è possibile perché 128 è maggiore di 117 e quindi scriviamo uno 0 in prima posizione a sinistra.

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Byte | | | | | | | |
| 0 | 1 | | | | | | |

Ora proviamo con il secondo peso il 64.

117 - 64 è possibile e otteniamo **53**, quindi scriviamo un 1 sotto il peso 64.

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Byte | | | | | | | |
| 0 | 1 | 1 | | | | | |

Ora usiamo il risultato della sottrazione per il peso 32.

53 - 32 è possibile e otteniamo **21**.

Scriviamo un 1 sotto il peso 32.

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Byte | | | | | | | |
| 0 | 1 | 1 | 1 | | | | |

Con il numero **21** proviamo con il prossimo peso che è il 16.

21 - 16 è possibile e otteniamo **5**.

Scriviamo 1 sotto il peso 16.

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Byte | | | | | | | |
| 0 | 1 | 1 | 1 | 0 | | | |

Usiamo il numero che abbiamo ottenuto: il 5.

5 - 8 non è possibile quindi scriviamo uno zero sotto al peso 8.

L' ELETTRONICA DIGITALE:

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Byte | | | | | | | |
| 0 | 1 | 1 | 1 | 0 | 1 | | |

Proviamo con il peso 4:

5 - 4 è possibile e otteniamo un **1**.

Scriviamo 1 sotto il peso 4.

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Byte | | | | | | | |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | |

Sottraiamo il nuovo peso a 1.

1 - 2 no è possibile e scriviamo uno **0** sotto al peso 2.

| Peso dei bit | | | | | | | |
|--------------|----|----|----|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| Byte | | | | | | | |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

Ultima sottrazione.

1 - 1 è possibile quindi scriviamo un **1** sotto all'ultimo peso.

Abbiamo ottenuto un numero bimario da un numero decimale.

Per vedere se è corretto sommiamo tutti i pesi che hanno un 1 sotto come abbiamo già fatto.

$$64 + 32 + 16 + 4 + 1 = 117$$

Per gestire numeri maggiori di 255 Arduino deve usare due Byte simulando un Byte da 16 bit che gli consente di usare numeri fino a 65535.

| Peso dei bit | | | | | | | | | | | | | | | |
|--------------|-------|------|------|------|------|-----|-----|-----|----|----|----|---|---|---|---|
| 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 16 Byte | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

L' ELETTRONICA DIGITALE:

Se sommiamo tutti i pesi di un Byte di 16 bit otteniamo:

$$32.768+16.384+8.192+4.096+1.024+512+256+128+64+32+16+8+4+2+1 = 65535.$$

Per memorizzare e gestire numeri più grandi di 65535 può usare al massimo 8 Byte da 8 bit arrivando ad un numero massimo di gestione di calcolo uguale a $1.7976931348623157 \times 10^{308}$.

Ora che conosciamo cosa è un bit e un Byte e come si legge e si scrive, possiamo imparare il linguaggio di Arduino: il C++. Però non si servirà a nulla se prima non impariamo a pensare come un programmatore, a pensare come una macchina.

PROGRAMMAZIONE:

Se in questo momento vi chiedessi che cosa state facendo, molti di voi che state leggendo questo manuale mi rispondereste: "Non sto facendo nulla, leggo il manuale."

Bene, non esiste peggior risposta. Primo non è vero che non state facendo nulla dal momento che state leggendo questo manuale, secondo non state solo leggendo; state respirando, un sensore misura il biossido di carbonio nel sangue e ordina una nuova respirazione, un altro sensore sta misurando l'orizzonte e vi impedisce di cadere dalla sedia, un altro sensore misura il livello di urina nella vescica e vi avvisa se è ora di andare al bagno e moltissime altre funzioni e cose che state facendo in questo momento.

Quindi non è vero che state solo leggendo un manuale.

Così è un programma che fa muovere una macchina, quando guardate una macchina ferma non è vero che non sta facendo nulla, realmente sta eseguendo molteplici istruzioni che come risultato danno la macchina ferma.

Un micro controllore o un micro processore non fanno nulla solo quando sono spenti, ugualmente per noi, facciamo nulla solo quando siamo morti.

Prima di scrivere un programma dobbiamo avere le idee chiare su quello che vogliamo ottenere altrimenti facciamo solo confusione.

Come primo programma facciamo lampeggiare un LED connesso a una uscita digitale di Arduino Uno. Un programma molto semplice però non stupido. Un programma semplice è solo una parte di un programma complesso perché un programma complesso è formato da istruzioni semplici.

Se ora vi chiedessi come si fa a far lampeggiare un LED quelli che avevano risposto "Non sto facendo nulla sto leggendo..." mi diranno: "Accendo e spengo il LED".

Non è corretto. Il ragionamento logico è:

- Accendo il LED per il tempo che voglio
- Spengo il LED per il tempo che voglio
- Ricomincio dall'inizio

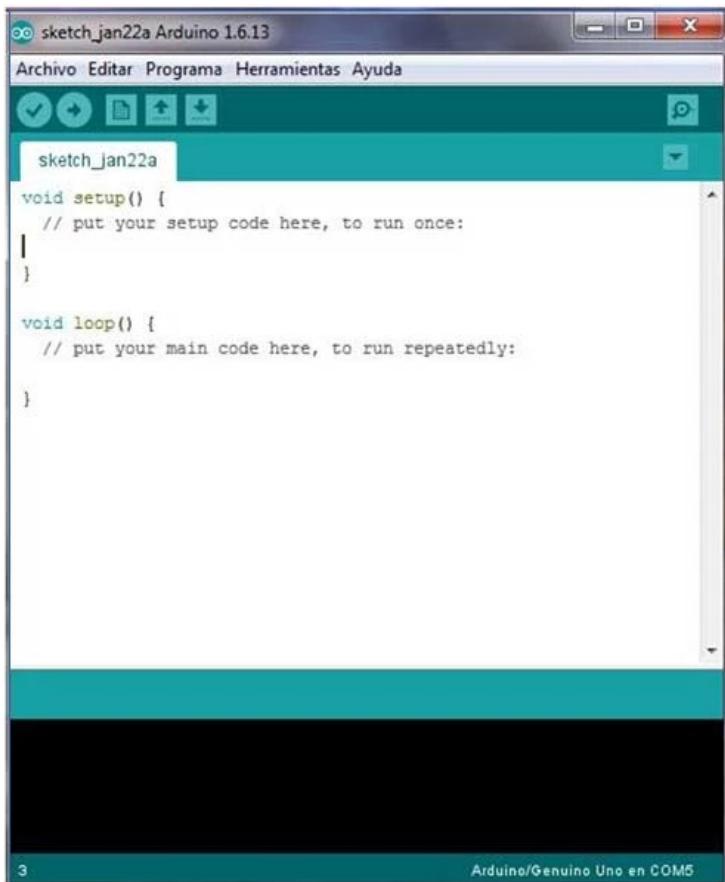
PROGRAMMAZIONE:

Il pensiero logico non è difficile serve solo un po' di esercizio. Una volta appreso vi porterà sempre un passo avanti agli altri, nella scuola, nel lavoro e nella vita.

Prima di scrivere il programma per fare lampeggiare il LED dobbiamo installare l'IDE sul nostro computer. Il software è gratis e si può scaricarlo dal sito ufficiale arduino.cc. Per ottenerlo è sufficiente scrivere in Google la chiave di ricerca "arduino download".

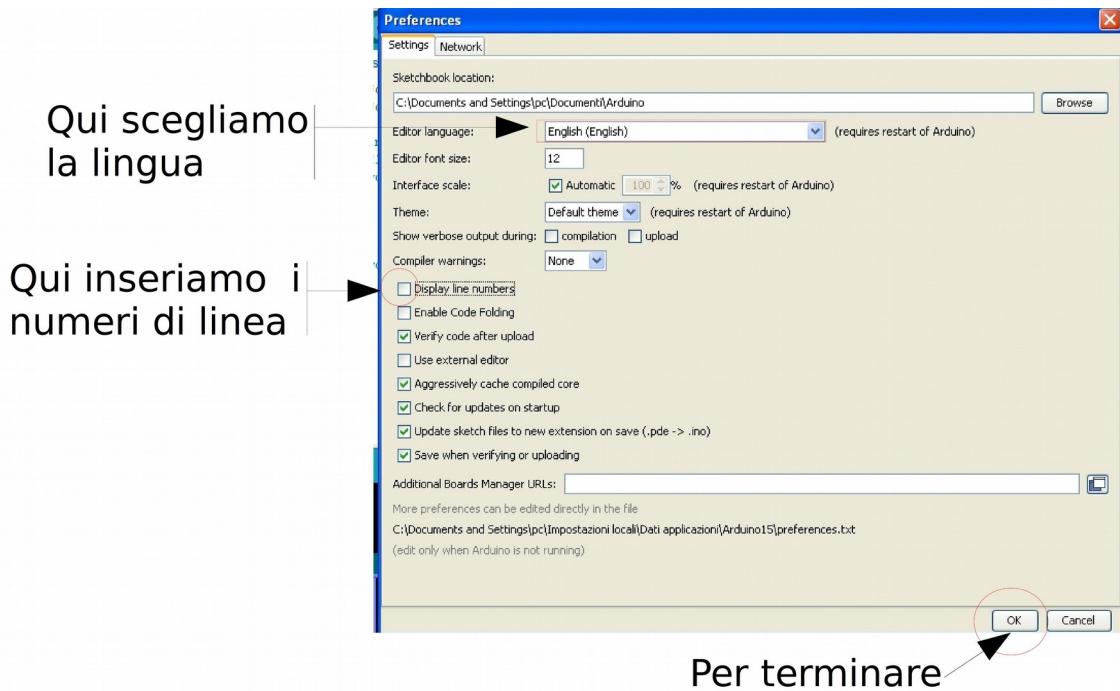
Sul sito si trovano tutte le istruzioni di installazione quindi salto questo passo.

La prima cosa da fare è inserire i numeri di linea e la lingua. Avviamo il programma facendo doppio clic sull'icona del programma e dopo pochi secondi avremo di fronte una finestra come questa:



Come prima cosa dobbiamo andare alla finestra delle preferenze per mettere i numeri di linea e la lingua in italiano. Sulla tastiera pigiamo il tasto "Ctrl" e, senza rilasciarlo, la virgola.

PROGRAMMAZIONE:



Chiudiamo l'IDE pigiando sulla tastiera i tasti "Ctrl" e, senza rilasciarlo, la "q". Riavviamolo e ora l'IDE si presenta con i numeri di linea ed in italiano.

Possiamo scrivere il primo **Sketch**, è così che si chiamano i programmi di Arduino, però prima ci serve schiarirci le idee.

Il primo Sketch è un programma che fa lampeggiare un LED.

Come scrisse per far lampeggiare un LED dobbiamo:

Accendere il LED per il tempo che vogliamo, spegnerlo per il tempo che vogliamo e ricominciare.

Scriviamolo.

Quando si apre la finestra del IDE si possono vedere due **funzioni** il "**void setup()**" e il "**void loop()**" che rappresentano la struttura base del programma.

Alla funzione "setup()" segue la parentesi graffa aperta che contiene tutte le istruzioni che deve eseguire la funzione. La funzione "setup()" si esegue solo all'accensione di Arduino o dopo un reset. Nel "setup()" si scrivono tutte le istruzioni di settaggio del micro controllore.

PROGRAMMAZIONE:



```
sketch_sep14a

void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:
}
```

La funzione "loop()" contiene tutte le istruzioni che il micro controllore deve eseguire. Si chiama "loop" perché non ha fine. Quando esegue l'ultima istruzione ricomincia.

Noi vogliamo far lampeggiare un LED. Come prima cosa dobbiamo decidere dove collegarlo.



Possiamo scegliere tra le 13 uscite/entrate digitali. Per questo esercizio useremo la numero 2 alla quale collegheremo il LED.

Nella funzione "setup()" dobbiamo dire al micro controller che la connessione 2 deve essere una uscita in questo modo: "**pinMode (2, OUTPUT);**"

PROGRAMMAZIONE:



Quando scriviamo una istruzione l'IDE la riconosce e la scrive in rosso.

Il comando “pinMode” è composto da due parole “pin” e “Mode” (si chiama pin la connessione).

Nel linguaggio C++ la prima parola si scrive minuscola e la seconda con la prima lettera maiuscola, questo modo di scrivere si chiama “a cammello”.

Dopo la istruzione, nella parentesi rotonda si mette come primo numero la connessione, il pin, e poi separata da una virgola il modo; “(2, OUTPUT)” dove “2” è il pin che vogliamo configurare e “OUTPUT” configura il pin come uscita.

Tutte le istruzioni terminano col punto e virgola.

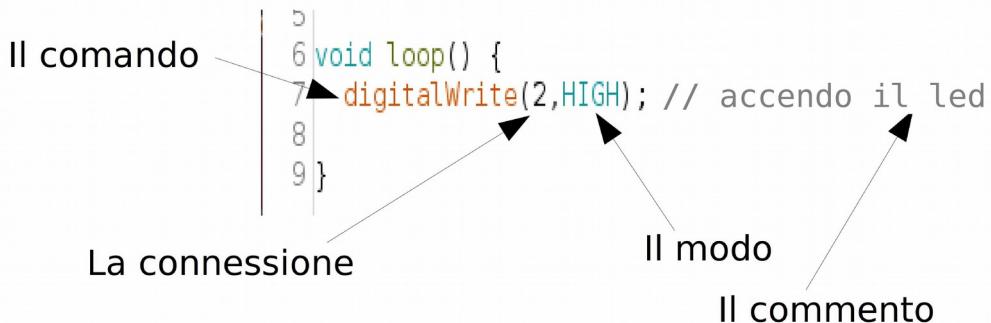
A seguito di una istruzione è buona cosa mettere sempre un commento per noi e soprattutto per chi leggerà il nostro programma perché se non lo facciamo a distanza di tempo non sapremo più cosa avevamo pensato quel giorno.

Per inserire un commento al termine della istruzione si mettono due barre “//” e l'IDE non considererà il nostro commento.

Il “void loop()” è una funzione e ha le sue istruzioni chiuse tra le parentesi graffe.

Le parentesi graffe racchiudono sempre un gruppo di istruzioni.

PROGRAMMAZIONE:



La prima istruzione che scriviamo nel "loop()" è l'istruzione che accende il LED : " **digitalWrite** (2, **HIGH**); " .

" **digitalWrite** " significa " scrivi la conessione digitale " e siccome è una istruzione l'IDE la scrive in rosso.

Tutte le connessioni digitali di arduino sono bit di un Byte e pertanto possono avere solo due stati logici "1" o "0", "ALT0" o "BASS0".

L'uno è la tensione positiva del micro controllore che nel caso del Atmega328 è di 5 Volt e zero è la tensione negativa del micro controllore uguale a 0 Volt, quindi se mettiamo alta (HIGH) l'uscita abbiamo 5 Volt e il LED si accenderà.

Dopo aver scritto il comando, nella parentesi rotonda inseriamo la connessione e il modo (2, HIGH), dove il 2 è la connessione e HIGH è il modo che in questo caso l'IDE lo scrive in blu perché è uno stato logico. Mettiamo il punto e virgola e di seguito il commento "// accendo il LED" in modo che sappiamo quello che abbiamo fatto.

Ora ci serve che il LED resti acceso per il tempo che vogliamo, per esempio un secondo.

```

5
6 void loop() {
7   digitalWrite(2,HIGH); // accendo il led
8   delay(1000);         // aspetto un secondo
9
10 }

```

La funzione
Il valore
Il commento

PROGRAMMAZIONE:

Scriviamo: “ **delay** (1000); ”

“ **delay** ” potrebbe sembrare una istruzione ma in realtà è una funzione che ferma il micro controllore per il valore del tempo scritto in parentesi. Il tempo nelle parentesi è espresso in millisecondi perciò per fermare il micro controllore per un secondo dobbiamo scrivere 1000.

1000 milli secondi = 1 secondo

ora possiamo aggiungere il commento: “// aspetto un secondo”

Fino ad ora abbiamo acceso il LED e lo abbiamo mantenuto acceso per un secondo.

```

5
6 void loop() {
7   digitalWrite(2,HIGH); // accendo il led
8   delay(1000);         // aspetto un secondo
9   digitalWrite(2,LOW);  // spegno il led
10
11 }

```

Il comando
La connessione
Il modo
Il commento

Per spegnere il LED usiamo lo stesso comando che abbiamo usato per accenderlo portando BASSA l'uscita in modo che la connessione si porti a zero Volt e il LED senza corrente si spenga.

“ **digitalWrite** (2, **LOW**); ” dove “ **digitalWrite** ” è l'istruzione che

scrive l'uscita logica "2" è l'uscita che andiamo a scrivere e "LOW" è lo stato logico uguale a zero.

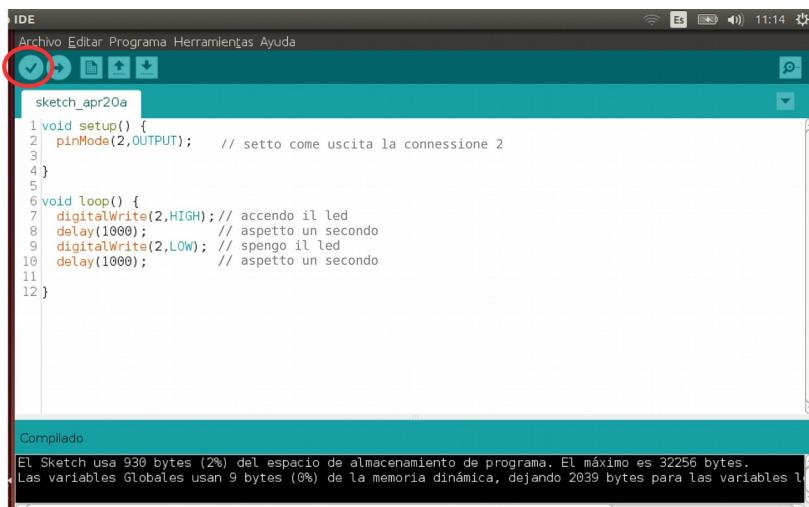
Nel commento scriviamo "// spengo il LED".

Ora dobbiamo decidere per quanto tempo tenere spento il LED. Per ottenere un lampeggio regolare spegniamolo per un secondo. Scriviamo un'altra volta " delay(1000);" e dopo il commento "// aspetto un secondo".

PROGRAMMAZIONE:

Ora si può capire l'importanza del commentario. Guardando il commentario possiamo leggere il programma come un libro e capire come sta funzionando.

Il nostro primo programma è finito ora possiamo vedere se è corretto. Per verificare se ci sono errori clicchiamo il primo bottone a sinistra nella finestra dell'IDE.



Prima di fare la verifica l'IDE salva il programma nella cartella che abbiamo scelto nelle preferenze o in quella di default. Se non ci sono errori nella parte bassa della finestra compare una scritta bianca che ci comunica quanto spazio occupa il programma nella memoria di Arduino.

In caso di errori compare una scritta in rosso che comunica il tipo di errore e la linea dove l'IDE ha trovato l'errore si evidenzia.

Prima di collegare il LED all'uscita digitale di Arduino dobbiamo sapere che l'uscita digitale di Arduino ha 5 Volt e una corrente di 40 mA (milli Amper) però tutto il micro controllore ha una corrente massima di 200 mA (milli Amper) che vuol dire che se assorbiamo 40 mA da più di 5 uscite bruciamo il micro controller quindi mettiamo sempre una resistenza per abbassare la corrente in uscita e perché un LED collegato a 5 Volt e 40 mA si brucerebbe.

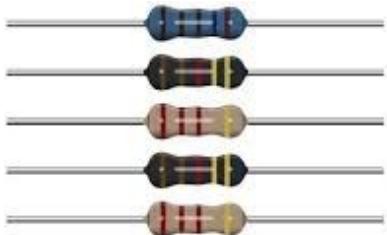
LE RESISTENZE: unità di misura in Ohm

L'unità di misura della resistenza elettrica è in Ohm indicata dalla lettera greca omega Ω .

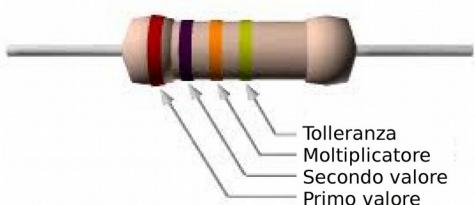
Un Ohm corrisponde alla resistenza che incontrano gli elettroni a muoversi in una colonna di mercurio alta 1063 mm dal peso di 14452 grammi a una temperatura di zero gradi Celsius.

Una resistenza elettrica oltre al suo valore espresso in Ohm ha un altro valore: la potenza massima che può dissipare senza bruciarsi.

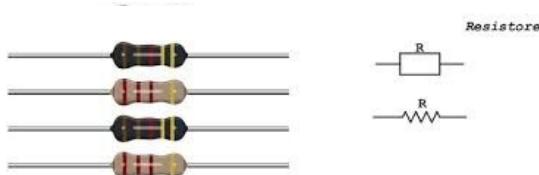
In commercio le resistenze in polvere di graffite hanno potenze di $1/8$, $\frac{1}{4}$, $\frac{1}{2}$, 1 y 2 Watt. Resistenze in grado di dissipare potenze superiori come 3, 5, 10, 20 y 30 Watt sono costruite con il filo al nichel cromo.



Sul corpo della resistenza ci sono degli anelli colorati che indicano il valore della resistenza. Questi anelli indicano il valore in Ohm e la tolleranza di precisione del valore dichiarato.



Ci sono dieci colori che corrispondono ai numeri da 0 a 9. Si legge da sinistra a destra iniziando dall'anello più vicino alla connessione elettrica.



Simbolo elettrico della resistenza.

LE RESISTENZE: unità di misura in Ohm

Resistenze con 4 anelli

| colore | 1° anello | 2° anello | 3° anello | 4° anello | moltiplicatore | tolleranza |
|-----------|-----------|-----------|------------------|-----------|----------------|------------|
| | 1° cifra | 2° cifra | | | | |
| nero | - | 0 | x1 | - | | |
| marrone | 1 | 1 | x10 | - | | |
| rosso | 2 | 2 | x100 | - | | |
| arancione | 3 | 3 | x1000 (1KΩ) | - | | |
| giallo | 4 | 4 | x10000 (10KΩ) | - | | |
| verde | 5 | 5 | x100000 (100KΩ) | - | | |
| blu | 6 | 6 | x1000000 (1MΩ) | - | | |
| viola | 7 | 7 | x10000000 (10MΩ) | - | | |
| grigio | 8 | 8 | x100000000 | - | | |
| bianco | 9 | 9 | - | 5% | | |
| oro | - | - | :10 | 10% | | |
| argento | - | - | :100 | 20% | | |

Resistenze con 5 anelli

| colore | 1° anello | 2° anello | 3° anello | 4° anello | 5° anello | moltiplicatore | tolleranza |
|-----------|-----------|-----------|-----------|-----------------|-----------|----------------|------------|
| | 1° cifra | 2° cifra | 3° cifra | 4° anello | 5° anello | | |
| nero | 0 | 0 | 0 | x1 | | | ±1% |
| marrone | 1 | 1 | 1 | x10 | | | ±2% |
| rosso | 2 | 2 | 2 | x100 | | | |
| arancione | 3 | 3 | 3 | x1000 (1KΩ) | | | |
| giallo | 4 | 4 | 4 | x10000 (10KΩ) | | | |
| verde | 5 | 5 | 5 | x100000 (100KΩ) | | | ±0,5% |
| blu | 6 | 6 | 6 | x1000000 (1MΩ) | | | ±0,25% |
| viola | 7 | 7 | 7 | x10000000(10MΩ) | | | ±0,1% |
| grigio | 8 | 8 | 8 | x100000000 | | | ±0,025% |
| bianco | 9 | 9 | 9 | x1000000000 | | | |
| oro | - | - | - | :10 | | | ±5% |
| argento | - | - | - | :100 | | | ±10% |

È molto semplice leggere il valore di una resistenza: il primo colore rappresenta la prima cifra del valore, il secondo la seconda cifra e il terzo colore il numero di zeri che seguono.

Se abbiamo una resistenza con quattro anelli e i primi tre sono rossi e il quarto è oro, il primo anello sarà un 2, il secondo anello sarà un 2 e il terzo anello sarà due zeri che seguono al numero quindi il suo valore sarà di 2200 Ω (Ohm). L'anello oro indica una tolleranza del 5%.

Se abbiamo una resistenza con cinque anelli: rosso, rosso, nero, marrone e oro il suo valore sarà: rosso=2, rosso=2, nero=0 e il marrone che aggiunge uno zero quindi anche questa sarà da 2200 Ω (Ohm) con una tolleranza del 5%.

Se il terzo anello è oro o argento il valore dei primi numeri si moltiplica per 0.1 per l'anello oro e per 0.01 per l'anello argento. Moltiplicare per 0.1 e per 0.01 è come dividere per 10 o per 100.

$$22 * 0.1 = 2.2 \\ 22 * 0.01 = 0.22$$

$$22 / 10 = 2.2 \\ 22 / 100 = 0.22$$

Nel dubbio per leggere correttamente una resistenza ricordiamoci sempre che una resistenza non può mai iniziare con anelli di colore oro o argento.

Siccome non è possibile costruire resistenze con precisione assoluta, i costruttori raggruppano le resistenze per gruppi di tolleranza dichiarata dall'ultimo anello. Per esempio se abbiamo una resistenza da 1000 Ω con una tolleranza del 5% il suo valore sarà compreso tra 950 e 1050 Ohm.

LE RESISTENZE: unità di misura in Ohm

I valori delle resistenze sono standard e hanno i suoi multipli:

kilo Ohm (KΩ) = * 1,000 Ohm

Mega Ohm (MΩ) = * 1,000,000 Ohm

Le resistenze hanno valori standard.

| 1 Ω | | | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1,2 Ω | 1,5 Ω | 1,8 Ω | 2,2 Ω | 2,7 Ω | 3,3 Ω | 3,9 Ω | 4,7 Ω | 5,6 Ω | 6,8 Ω | 8,2 Ω | 10 Ω |
| 12 Ω | 15 Ω | 18 Ω | 22 Ω | 27 Ω | 33 Ω | 39 Ω | 47 Ω | 56 Ω | 68 Ω | 82 Ω | 100 Ω |
| 120 Ω | 150 Ω | 180 Ω | 220 Ω | 270 Ω | 330 Ω | 390 Ω | 470 Ω | 560 Ω | 680 Ω | 820 Ω | 1K Ω |
| 1,2K Ω | 1,5K Ω | 1,8K Ω | 2,2K Ω | 2,7K Ω | 3,3K Ω | 3,9K Ω | 4,7K Ω | 5,6K Ω | 6,8K Ω | 8,2K Ω | 10K Ω |
| 12K Ω | 15K Ω | 18K Ω | 22K Ω | 27K Ω | 33K Ω | 39K Ω | 47K Ω | 56K Ω | 68K Ω | 82K Ω | 100K Ω |
| 120K Ω | 150K Ω | 180K Ω | 220K Ω | 270K Ω | 330K Ω | 390K Ω | 470K Ω | 560K Ω | 680K Ω | 820K Ω | 1M Ω |
| 1,2M Ω | 1,5M Ω | 1,8M Ω | 2,2M Ω | 2,7M Ω | 3,3M Ω | 3,9M Ω | 4,7M Ω | 5,6M Ω | 6,8M Ω | 8,2M Ω | 10M Ω |

È semplice impararli perché sono solo 11 valori che si ripetono:

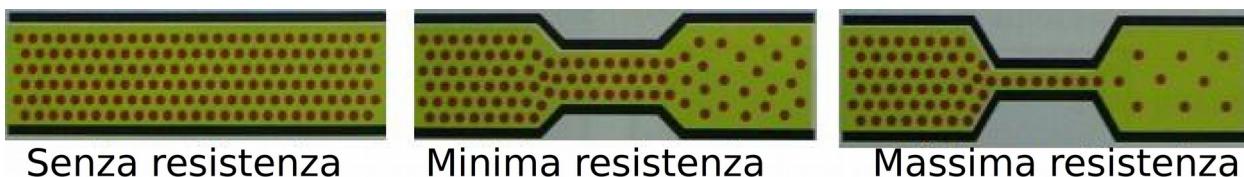
1 – 1.2 – 1.8 – 2.2 – 2.7 – 3.3 – 3.9 – 4.7 – 5.6 -6.8 -8.2

10 -12 -18 -22 -27 -33 -39 - 47 -56 - 68 – 82

come si può vedere in tabella.

LE RESISTENZE: a cosa servono

Una resistenza posta in serie ad un circuito provoca una caduta di corrente perché frena il passaggio degli elettroni. Se ad un conduttore capace di far passare un certo numero di elettroni mettiamo in serie una resistenza è intuitivo che il flusso sarà rallentato. Possiamo paragonare la resistenza elettrica ad una strozzatura in un tubo idraulico.

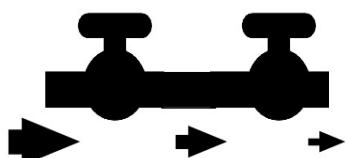


Se il tubo non presenta strozzature l'acqua corre liberamente, se lo strozziamo leggermente l'acqua ridurrà la pressione e se lo stringiamo ulteriormente l'acqua incontrerà una resistenza maggiore rallentando ancora.

Quando una corrente elettrica incontra una resistenza che impedisce agli elettroni di correre liberamente questi generano calore. Questa proprietà si chiama effetto Joule e grazie a questo possiamo produrre calore con la corrente elettrica per vari impieghi come il saldatore elettrico, l'asciuga capelli, la stufa elettrica, la cucina elettrica e molto altro.

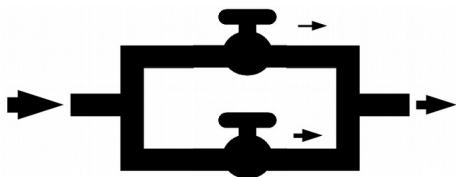
LE RESISTENZE: in serie e in parallelo

Collegare due resistenze in serie vuol dire metterle una dietro l'altra.



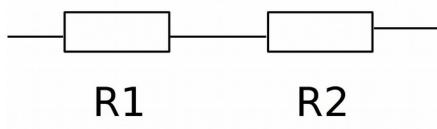
Possiamo paragonare due resistenze in serie a due rubinetti uno dopo l'altro. In questo caso l'acqua è frenata da entrambi i rubinetti.

Collegare due resistenze in parallelo vuol dire metterle accoppiate.



Possiamo paragonare due resistenze in parallelo a due rubinetti affiancati come in figura. In questo caso il flusso dei due rubinetti si somma.

$$\text{Ohm} = R_1 + R_2$$



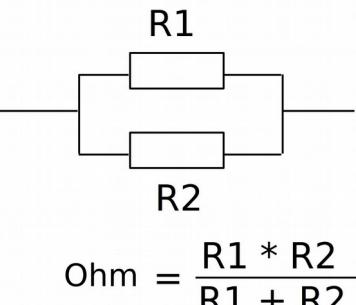
Collegando in serie due resistenze il valore risultante sarà la somma dei valori delle due resistenze.

$$\text{Ohm} = R_1 + R_2$$

$$1,200 + 1.500 = 2,700 \text{ Ohm}$$

Collegando due resistenze in parallelo il valore risultante sarà inferiore al valore della resistenza più bassa.

$$\text{Ohm} = (R_1 * R_2) / (R_1 + R_2)$$

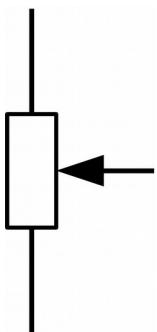


$$(1,200 * 1,500) / (1,200 + 1,500) = 666.66 \text{ Ohmios}$$

La corrente elettrica come l'acqua è pigra e va sempre dove fa meno fatica.

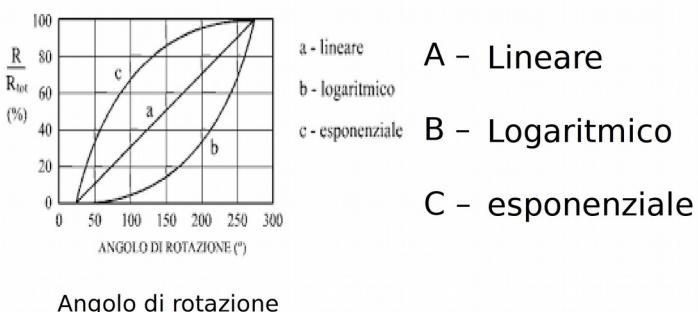
LE RESISTENZE: i Potenziometri:

Quando in un circuito elettronico ci serve una resistenza capace di cambiare il suo valore ohmmico da zero al suo massimo dobbiamo usare un trimmer o un potenziometro.



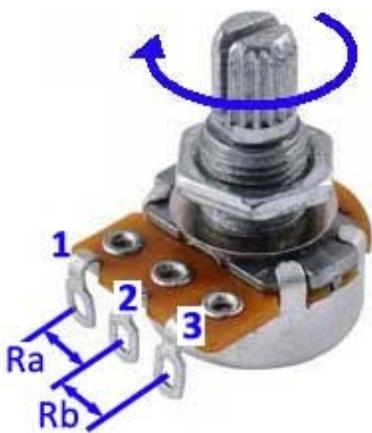
Nello schema elettrico il simbolo del potenziometro o del trimmer è uguale a quello della resistenza con una freccia che rappresenta il cursore. Il valore del potenziometro è indicato con tre numeri: il primo è la prima cifra, il secondo è la seconda e il terzo è il numero di zeri che seguono.

$103 = 10,000 \text{ ohm} = 10\text{K}\Omega$ (Kilo Ohm). I potenziometri si dividono in logaritmici e lineari. Il potenziometro lineare varia la sua resistenza in modo proporzionale mentre quello logaritmico segue una curva esponenziale.



Nel potenziometro e nel trimmer la connessione 2 si muove sulla pista resistiva dalla connessione 1 alla 3 e viceversa.

LE RESISTENZE: i Potenziometri:



Il potenziometro logaritmico è utilizzato principalmente negli amplificatori audio come controllo di volume perché il nostro orecchio è un senso logaritmico. Un esempio: se giriamo a metà corsa l'alberino di un potenziometro lineare da 10 kilo Ohm (10000 Ohm) la resistenza che misuriamo tra la connessione 1 e 2 (R_a) e quella tra la connessione 2 e 3 (R_b) scopriamo che il valore risulta uguale alla metà del suo valore $R_a = 5 \text{ k}\Omega$ e $R_b = 5 \text{ k}\Omega$.

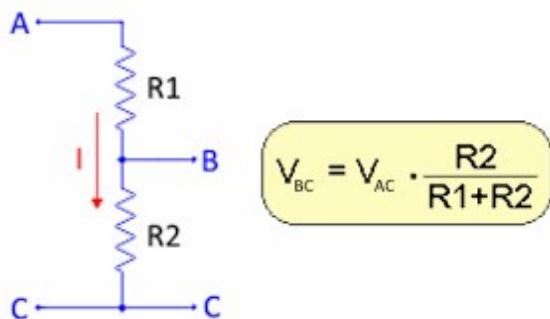
Se giriamo l'alberino a tre quarti il suo valore sarà $R_a = 7500 \text{ Ohm}$ e $R_b = 2500 \text{ Ohm}$.

Se mettiamo a metà corsa un potenziometro logaritmico da 10000 Ohm scopriremo che il valore di R_a sarà di 1000 Ohm e R_b di 9000 Ohm. E se lo mettiamo a tre quarti $R_a=6500 \text{ Ohm}$ e $R_b=3500 \text{ Ohm}$. In effetti il nostro orecchio sente un raddoppio della potenza sonora se quadruplica la potenza.

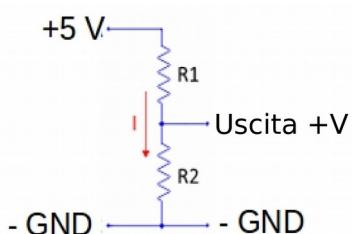
Per distinguere un potenziometro lineare da uno logaritmico nella sigla troviamo una lettera: A per lineare e B per logaritmico.

Il mercato asiatico usa un altro stile: il valore si scrive in Ohm ($10K = 10 \text{ Kilo Ohm} = 10000 \text{ Ohm}$) e la lettera A che sta per audio indica il potenziometro logaritmico e la B indica il potenziometro lineare. Purtroppo non è una regola precisa, per sicurezza è meglio basarsi sul paese di origine o provare con un multmetro.

LE RESISTENZE: partitore di tensione



Un partitore di tensione si può comparare al miscelatore del lavabo. Il miscelatore ha due ingressi: uno per l'acqua fredda e uno per quella calda. Se mettiamo al centro la leva del miscelatore esce acqua tiepida formata da un 50% di acqua calda e un 50% di acqua fredda.



Il partitore di tensione è uguale. Se la resistenza R1 è uguale a R2 in uscita avremo il 50% della tensione in entrata secondo la formula:

$$V(\text{uscita}) = V(\text{ingresso}) * \frac{R_2}{R_1+R_2}$$

V (entrata) è uguale a 5 Volt e R1 e R2 immaginiamole da 10 Kilo Ohmios.

$$5V * \frac{10000}{10000 + 10000}$$

LE RESISTENZE: partitore di tensione

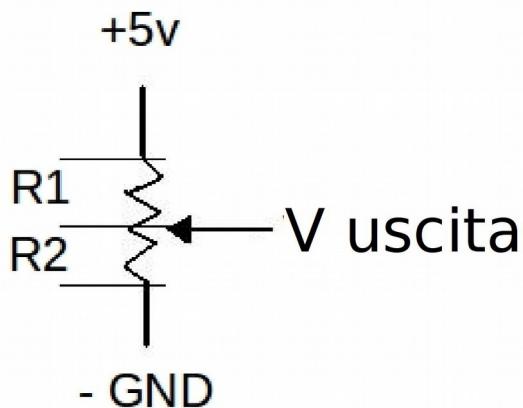
In questo caso sommiamo il valore di R1 e R2 (10000 + 10000) e otteniamo 20000 Ohm poi dividiamo il valore di R2 (10000 Ohm) per la somma di R1 + R2.

$$5V * \frac{10000}{20000}$$

$$10000 / 20000 = 0.5$$

Ora moltiplichiamo il valore della tensione in ingresso per il risultato:

$$5V * 0.5 = 2.5V$$



Un potenziometro può essere un partitore di tensione variabile.

LE FOTORESISTENZE:



Le fotoresistenze sono componenti elettronici molto sensibili alla luce che possono variare la propria resistenza in modo inversamente proporzionale alla luce che ricevono.

Nell'oscurità la fotoresistenza ha un valore di circa 1 MΩ (1 Mega Ohm – 1000000 Ohm), se riceve un po' di luce il suo valore si abbassa a 400 KΩ (400 Kilo Ohm – 400000 Ohm) e con una luce molto forte scende a pochi Ohm.

La fotoresistenza si usa quando occorre creare sistemi automatici che usano la luce come accendere una lampada quando arriva la notte e molte altre applicazioni.

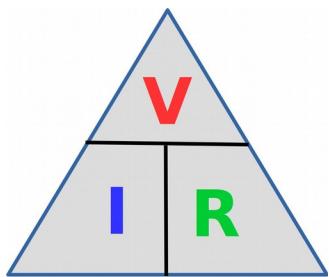
La fotoresistenza ha una corrente molto bassa e non può comandare direttamente una lampadina, ha bisogno di un circuito dedicato.

LA LEGGE DI OHM:

La legge di Ohm si usa per determinare la relazione tra tensione, corrente e resistenza in un circuito elettrico.

La legge di Ohm riceve il nome in onore del fisico tedesco Georg Ohm (1789-1854).

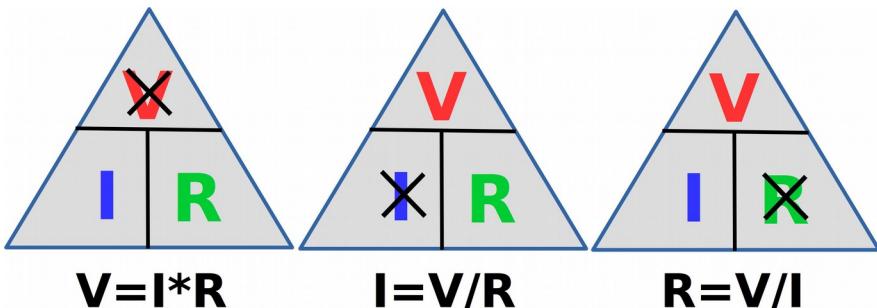
Si presenta in forma esplicita con la formula:



$$\text{Volt} = \text{Amper} * \text{Ohm}, \text{ o } V = A * \Omega.$$

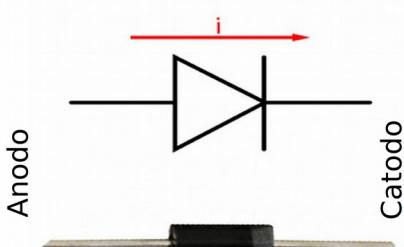
Osservando il triangolo della legge di Ohm è facile capire come ottenere tutte le formule necessarie per trovare il dato che ci interessa.

È sufficiente eliminare il dato che ci serve sapere dal triangolo.



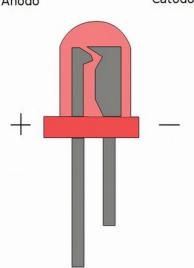
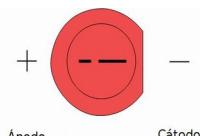
In questo modo possiamo conoscere il valore della tensione, della corrente e della resistenza.

I DIODI LED:



I LED sono diodi. LED è l'acronimo di Light Emitting Diode che significa Diodo a Emissione Luminosa. I diodi hanno due terminazioni che si chiamano Catodo e Anodo. Nel diodo la corrente può circolare solo in un senso. Tutti i diodi hanno un anello stampato sul loro corpo che indica il terminale negativo o Catodo per differenziarlo da quello positivo o Anodo. I diodi si diffenziano per tensione, corrente e velocità operativa.

I diodi trovano svariate applicazioni in elettronica che approfondiremo più avanti, ora parlerò solo dei LED.



I LED possono emettere luce colorata o bianca. Vengono prodotti di un solo colore, con due colori e RGB (Red, Green, Blue) un LED che nel suo corpo ha tre LED uno di colore rosso, uno verde e uno blu. Questi tre colori possono generare la luce bianca e tutti i colori.

Possono avere misure e forme differenti. I LED si accendono solo se al terminale Anodo è collegato il positivo dell'alimentazione (+V, VCC) e al Catodo il negativo dell'alimentazione (-V, GND).

Il terminale Anodo si riconosce perché è il più lungo. Se il LED avesse i terminali tagliati è possibile riconoscere il Catodo guardando il LED da sopra, il lato del Catodo è piatto.

Un LED non si può collegare direttamente a una fonte di corrente o a una pila perché si brucerà immediatamente .

Per accendere un LED senza distruggerlo dobbiamo mettere in serie a uno dei due terminali una resistenza.



I DIODI LED:

Il LED esige una corrente da 0.015 a 0.017 Amper (15 o 17 milli Amper – mA). Ora capiamo perché non possiamo collegarlo direttamente ad una uscita digitale di Arduino che ha 40 mA.

Per collegare un LED a una uscita digitale di Arduino dobbiamo mettere in serie una resistenza. Per sapere il valore della resistenza possiamo riferirci alla legge di Ohm.

$$\text{Ohm} = V : A.$$

$$\text{Ohm} = (V - 1.5) / 0.016$$

dove:

Ohm è il valore della resistenza

V è la tensione in uscita di Arduino

1.5 è la caduta di tensione del LED

0.016 è la corrente in Amper che richiede il LED

$$(5 - 1.5) / 0.016 = 218.75$$

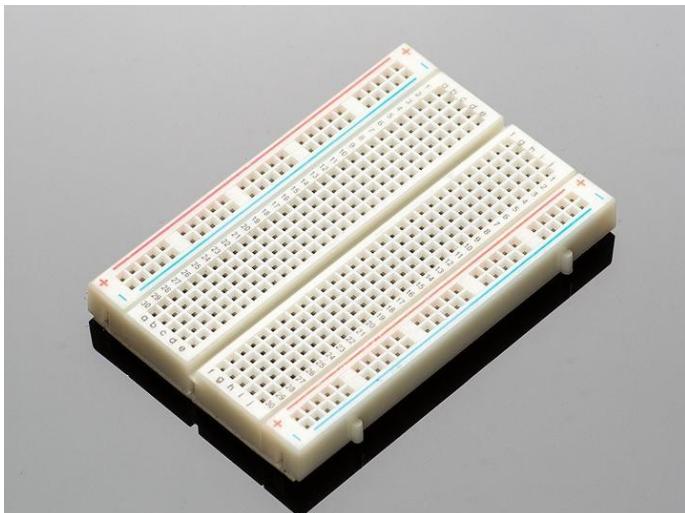
Il valore della resistenza 218.75 Ohm non esiste quindi scegliamo il valore commerciale più vicino **220 Ohmios**.

Facendo la prova al rovescio:

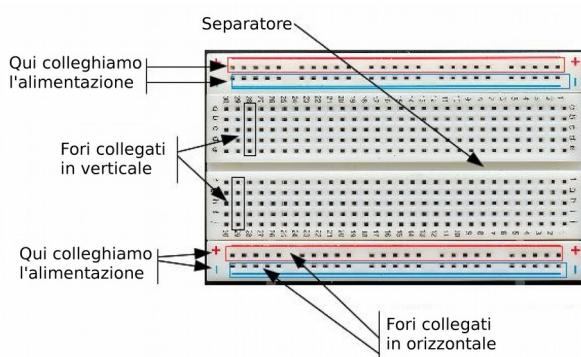
$$(5 - 1.5) / 220 = 0,015909 \text{ Amper}$$

poco meno di 16 milli Amper (mA) corretto!

LA BASETTA DI PROVA:

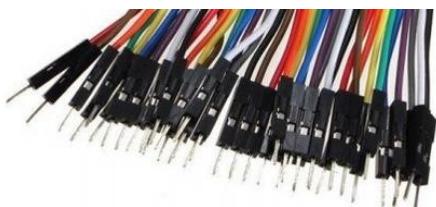


La scheda di prova (*Protoboard* o *Breadboard*) è una scheda che presenta molti fori con passo di un decimo di pollice (2.54 mm) collegati elettricamente tra loro per collegare componenti elettronici per testare i circuiti prima di sviluppare i circuiti stampati.



Come si può vedere in figura la basetta è divisa in due parti orizzontalmente. Le connessioni vicine al bordo sono dell'alimentazione e sono collegate in orizzontale, quelle centrali per l'inserzione dei componenti sono collegate in verticale.

Per fare i collegamenti si usano cavi specifici che hanno un puntale rigido che si chiamano Dupont.



Per le connessioni si possono anche usare spezzoni di cavi LAN classe 6 usati per le reti dei computer.

ARDUINO COLLEGARE UN LED:

Ora che sappiamo cosa sia la corrente elettrica, cosa sono le resistenze , i LED e come usare la basetta di prova possiamo collegare il LED ad Arduino.

Per farlo ci serve la basetta di prova, un LED e una resistenza da 220Ω . Per riconoscere la resistenza facciamo riferimento alla tavola dei colori.

Resistenze con 4 anelli

| colore | 1° anello | 2° anello | 3° anello | 4° anello | |
|-----------|-----------|-----------|---------------------------|------------|--|
| | 1° cifra | 2° cifra | moltiplicatore | tolleranza | |
| nero | | 0 | x1 | - | |
| marrone | 1 | 1 | x10 | - | |
| rosso | 2 | 2 | x100 | - | |
| arancione | 3 | 3 | x1000 ($1K\Omega$) | - | |
| giallo | 4 | 4 | x10000 ($10K\Omega$) | - | |
| verde | 5 | 5 | x100000 ($100K\Omega$) | - | |
| blu | 6 | 6 | x1000000 ($1M\Omega$) | - | |
| ciano | 7 | 7 | x10000000 ($10M\Omega$) | - | |
| grigio | 8 | 8 | x100000000 | - | |
| bianco | 9 | 9 | - | 5% | |
| oro | | | :10 | 10% | |
| argento | - | - | :100 | 20% | |

Resistenze con 5 anelli

| colore | 1° anello | 2° anello | 3° anello | 4° anello | 5° anello | |
|-----------|-----------|-----------|-----------|----------------------------|---------------|--|
| | 1° cifra | 2° cifra | 3° cifra | moltiplicatore | tolleranza | |
| nero | 0 | 0 | 0 | x1 | - | |
| marrone | 1 | 1 | 1 | x10 | $\pm 1\%$ | |
| rosso | 2 | 2 | 2 | x100 | $\pm 2\%$ | |
| arancione | 3 | 3 | 3 | x1000 ($1K\Omega$) | - | |
| giallo | 4 | 4 | 4 | x10000 ($10K\Omega$) | - | |
| verde | 5 | 5 | 5 | x100000 ($100K\Omega$) | $\pm 0,5\%$ | |
| blu | 6 | 6 | 6 | x1000000 ($1M\Omega$) | $\pm 0,25\%$ | |
| ciano | 7 | 7 | 7 | x100000000 ($10M\Omega$) | $\pm 0,1\%$ | |
| grigio | 8 | 8 | 8 | x1000000000 | $\pm 0,025\%$ | |
| bianco | 9 | 9 | 9 | x10000000000 | - | |
| oro | - | - | - | :10 | $\pm 5\%$ | |
| argento | - | - | - | :100 | $\pm 10\%$ | |

Se abbiamo una resistenza con quattro anelli:

il primo numero è 2 = rosso

il secondo numero è 2 = rosso

il numero degli zeri è 1 = marrone

ci serve quindi una resistenza che abbia i primi tre anelli rosso, rosso e marrone.

Se abbiamo una resistenza con cinque anelli:

il primo numero è 2 = rosso

il secondo numero è 2 = rosso

il terzo numero è 0 = nero

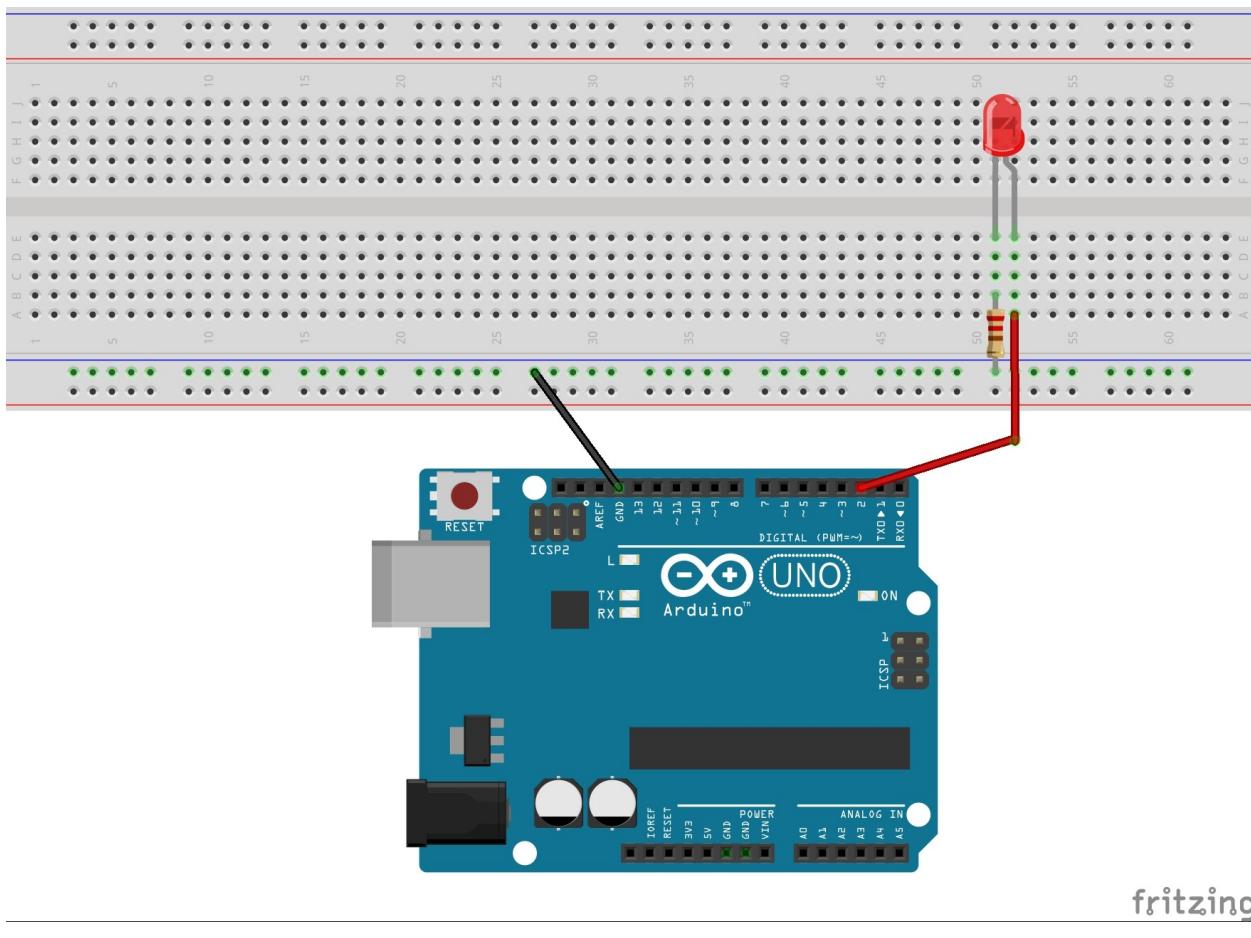
e il moltiplicatore è 1 perché $220 * 1 = 220$ quindi il quarto anello sarà nero.

Ci serve una resistenza che abbia i primi quattro anelli rosso, rosso, nero e nero.

Colleghiamo come in figura, usiamo un cavo nero per il negativo (0 V, -, GND) collegandolo alla barratura di alimentazione della piastra di prova evidenziata in azzurro con il segno “-” dove siamo più comodi perché tutti i fori sono collegati tra loro in orizzontale.

Arduino ha tre connessioni segnate con la scritta GND che sono connesse tra loro quindi possiamo usarne una qualunque ma per ora seguiamo lo schema di cablaggio. Montiamo il LED con l'anodo, il +V, a destra (il +V è la gamba più lunga). Colleghiamo con un filo rosso la connessione digitale 2 di Arduino (la terza perché i numeri iniziano da zero) e la collegiamo al +V del LED. La resistenza la montiamo tra il -V del LED e il -V, GND, della barratura della piastra di prova.

ARDUINO COLLEGARE UN LED:

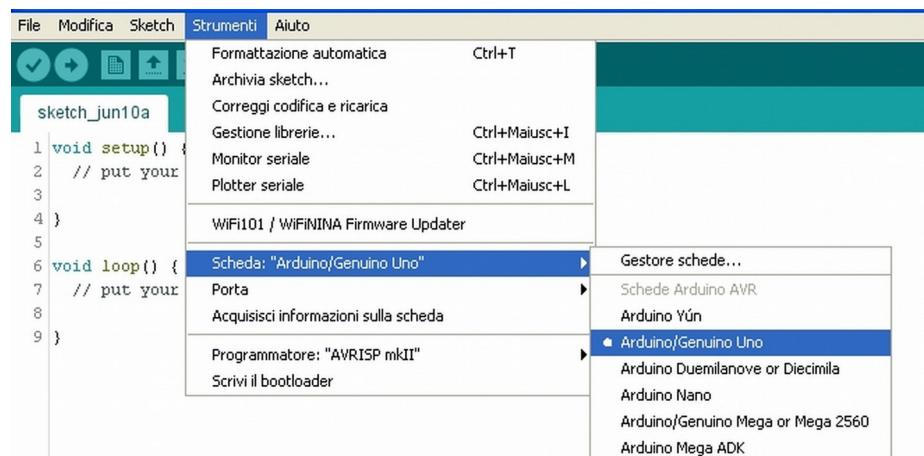


Ora possiamo collegare Arduino al computer con il cavo USB.

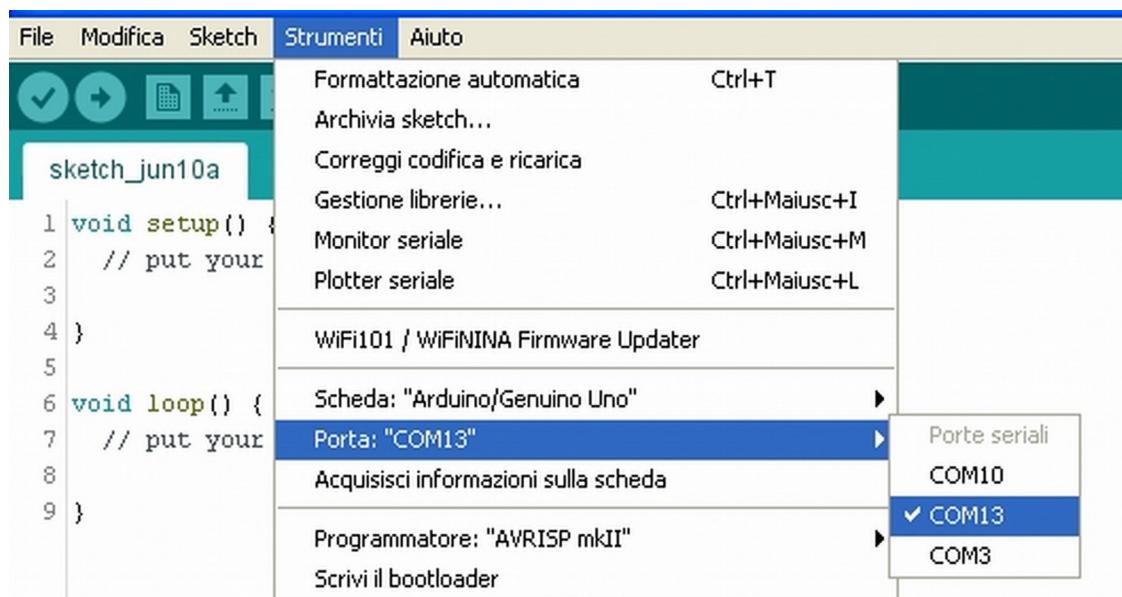
ARDUINO COLLEGARE UN LED:

Prima di caricare il programma dobbiamo comunicare all'IDE che tipo di Arduino abbiamo collegato e in quale porta.

Clicchiamo nella barra del menu "Strumenti" e scegliamo "Scheda", si apre una finestra nella quale possiamo scegliere "Arduino Uno".

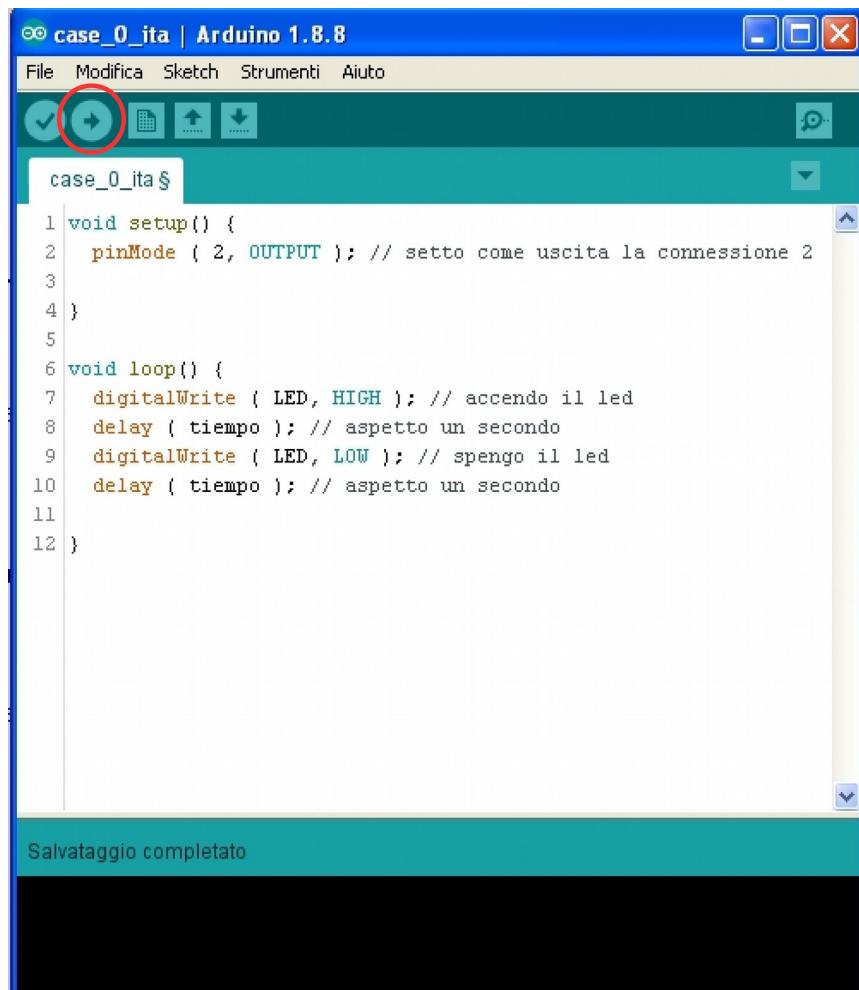


Sotto a "Scheda" troviamo "Porta" e clicchiamo la porta dove vediamo collegato Arduino.



ARDUINO COLLEGARE UN LED:

Per caricare il programma nella memoria del micro controllore clicchiamo sul secondo bottone in alto a sinistra.



L'IDE ora converte il codice in C++ in linguaggio macchina e lo carica nella memoria del micro controllore.

Se tutto è corretto possiamo vedere lampeggiare il LED. Se non si accende è perché lo abbiamo inserito al rovescio o abbiamo sbagliato un collegamento.

PROGRAMMAZIONE Le variabili:

Le variabili sono come dei cassetti o delle scatole dove andiamo a mettere i nostri dati (sia numeri che lettere).

A me piace immaginare la memoria del micro controllore come un magazzino con molte scatole.

Nelle scatole metto i miei dati però voglio ritrovarli quando mi servono e quindi metto una etichetta sulle scatole esattamente come si fa in un magazzino, Se in una scatola ritiro dei vasi sulla scatola scriverò "Vasi", se metto dei piatti scriverò "Piatti" in questo modo potrò sempre trovare quello che mi serve senza impazzire.

Il C++ non genera le variabili automaticamente, dobbiamo farlo noi.

Come già scrisse Arduino Uno ha una architettura a 8 bit e un Byte di 8 bit può contenere al massimo il numero 255. Per memorizzare un numero più grande dobbiamo usare più Byte. Nel nostro magazzino dobbiamo usare più scatole o una scatola più grande.

Come fa un buon magazziniere dobbiamo conoscere la quantità di cose che andremo ad immagazzinare per fare spazio e di contro sarebbe stupido sprecare spazio.

Per creare una variabile la sintassi è: `int vasi = 5;`

Dove "`int`" è il tipo di variabile che l'IDE scrive in blu,

"vasi" è il nome che voglio dare alla variabile.

"5" è il numero che metto nella variabile che ho chiamato "vasi".

Ora la variabile "vasi" contiene il numero "5".

Come un magazziniere gestisce scatole di diversa misura noi abbiamo variabili di diversa misura.

Le variabili sono:

- `byte` : può contenere un numero intero da 0 a 255, in un Byte di 8 bit
- `int` : (Integer=intero) può contenere un numero da -32768 a +32768 usa due Byte da 8 bit.
- `long` : può contenere numeri interi da -2147483647 a +2147483647 in 4 Byte da 8 bit.

PROGRAMMAZIONE Le variabili:

- **float** : può contenere numeri reali con decimali da 3.4028325E+38 a -3.4028325E+38 usando 4 Byte da 8 bit.
- **unsigned int** : può contenere un numero intero positivo da 0 a 65545 usando 2 Byte da 8 bit.
- **unsigned long** : può contenere un numero intero positivo da 0 a 4294967296 usando 4 Byte da 8 bit.
- **boolean** : può contenere solo un valore booleano: *true* (vero) o *false* (falso), on-off, HIGH-LOW
- **char** : può contenere un carattere ASCII in un Byte di 8 bit il carattere si memorizza come valore numerico ASCII.

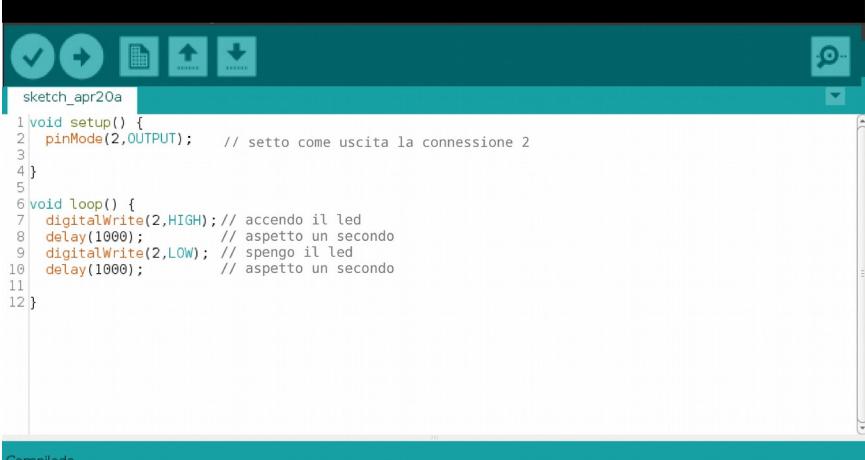
Queste sono i tipi di variabili (le casse del nostro magazzino) che possiamo usare.

Tutte le variabili possono essere di tipo **locale** o **globale**. Una variabile globale può essere usata in tutto il programma, una variabile locale solo all'interno delle parentesi graffe dove l'abbiamo creata.

Dobbiamo pensare bene al tipo di variabile che andiamo ad usare perché se provassi a memorizzare il numero "10.5" in una variabile di tipo "int" che non può contenere numeri con la virgola, si memorizzerà solo "10" e questo potrebbe generare errori di calcolo però è molto comodo usare questa proprietà per sbarazzarmi della parte decimale di un numero.

Prendiamo il programma che abbiamo scritto.

PROGRAMMAZIONE Le variabili:



```
sketch_apr20a
1 void setup() {
2   pinMode(2,OUTPUT); // setto come uscita la connessione 2
3 }
4
5 void loop() {
6   digitalWrite(2,HIGH); // accendo il led
7   delay(1000); // aspetto un secondo
8   digitalWrite(2,LOW); // spengo il led
9   delay(1000); // aspetto un secondo
10}
11
12}
```

Compilado

In questo programma abbiamo un tempo di attesa che è di 1000 millisecondi (1000 mS).

Ora andiamo a mettere questo valore in una variabile.

Come prima cosa dobbiamo dichiarare il tipo di variabile che vogliamo usare.

Guardando la finestra del IDE siamo portati a pensare che tutto il mondo di Arduino finisce con le funzioni "setup()" e "loop()" però non è così perché nella parte alta prima del "setup()" è dove andiamo a dichiarare le variabili. Una variabile creata sopra a tutto è una variabile globale che si può usare per tutto il programma.

Scriviamo sopra in alto prima del "setup()":

```
int tempo = 1000;
```

In questo modo abbiamo creato una variabile di tipo "int" perché il numero 1000 è un numero intero ed è troppo grande per una variabile di tipo "byte" che può contenere un numero massimo di 255.

Scriviamo il commento:

```
// la variabile "tempo" memorizza la frequenza del lampeggio
```

PROGRAMMAZIONE Le variabili:

```
1 int tempo = 1000; // memorizza la frequenza del lampeggio
2
3 void setup() {
4     pinMode ( 2, OUTPUT ); // setto come uscita la connessione 2
5
6 }
7
8 void loop() {
9     digitalWrite ( 2, HIGH ); // accendo il led
10    delay ( 1000 ); // aspetto un secondo
11    digitalWrite ( 2, LOW ); // spengo il led
12    delay ( 1000 ); // aspetto un secondo
13
14 }
```

Dentro le parentesi della funzione “**delay**” sostituiamo il numero 1000 con il nome della variabile.

```
1 int tempo = 1000; // memorizza la frequenza del lampeggio
2
3 void setup() {
4     pinMode ( 2, OUTPUT ); // setto come uscita la connessione 2
5
6 }
7
8 void loop() {
9     digitalWrite ( 2, HIGH ); // accendo il led
10    delay ( tempo ); // aspetto un secondo
11    digitalWrite ( 2, LOW ); // spengo il led
12    delay ( tempo ); // aspetto un secondo
13
14 }
```

La funzione “**delay()**” se vede una variabile dentro la parentesi legge il valore della variabile.

Se scriviamo un altro valore nella variabile “tempo” per esempio “500” vedremo il LED lampegiare più velocemente.

Int tempo = 500;

possiamo caricare il programma per provare.

PROGRAMMAZIONE Le variabili:

È molto comodo usare una variabile perché cambiando un solo valore cambiamo i due valori delle funzioni “**delay()**”.

Nel nostro primo Sketch (programma) per accendere e spegnere il LED abbiamo usato l'istruzione “ **digitalWrite** (2, **HIGH**); ” scrivendo il numero della connessione. Per un programma così semplice non è un problema ma se ci trovassimo a scrivere uno Sketch molto lungo e che usa molte uscite e entrate dovremo tenere a mente tutte le connessioni usate. Possiamo usare una variabile anche per i numeri delle connessioni e nel nostro caso chiamarla LED che ci ricorda che abbiamo collegato un LED.

DigitalWrite (LED, **HIGH**) ; **DigitalWrite** (LED, **LOW**) ;

in questo modo l'istruzione “ **digitalWrite** ” se vede una variabile legge il suo valore.

Quindi possiamo creare una seconda variabile sotto la prima e siccome il numero della connessione del LED è “2” e “2” è un numero minore di 255 possiamo usare una variabile di tipo “**byte**”.

byte LED = 2;

```
1 int tempo = 1000; // memorizza la frequenza del lampeggio
2 byte LED = 2; // la variabile LED memorizza la connessione del led
3
4 void setup() {
5   pinMode ( LED, OUTPUT ); // setto come uscita la connessione LED
6
7 }
8
9 void loop() {
10  digitalWrite ( LED, HIGH ); // accendo il led
11  delay ( tempo ); // aspetto un tempo
12  digitalWrite ( LED, LOW ); // spengo il led
13  delay ( tempo ); // aspetto un tempo
14
15 }
```

Facendo in questo modo se dovessimo cambiare la connessione del LED sarà sufficiente cambiare il valore alla variabile e non dovremo correggere tutto lo Sketch.

PROGRAMMAZIONE Le variabili:

Inoltre guardando le prime linee del programma si capisce subito che alla connessione 2 è collegato un LED.

Però in questo caso non è necessario usare una variabile che occupa spazio nella memoria di Arduino perché in una variabile, come dice il suo nome, mettiamo dei dati che ci serve cambiare nel corso del programma, in questo caso il numero della connessione del LED non cambia per tutto lo Sketch. Possiamo usare:

```
#define LED 3
```

che termina senza il punto e virgola e non ha neppure il simbolo dell'uguale perché è una istruzione speciale. Il “`#define`” comunica al IDE di cercare tutte le parole LED e di sostituirle con il numero “2”. In questo modo non usiamo spazio di memoria ottenendo lo stesso risultato.

L' IDE “`#define`” lo scrive in grigio.

```
1 #define LED 2 // è la connessione del led
2
3 int tempo = 1000; // memorizza la frequenza del lampeggio
4
5
6 void setup() {
7   pinMode ( LED, OUTPUT ); // setto come uscita la connessione LED
8
9 }
10
11 void loop() {
12   digitalWrite ( LED, HIGH ); // accendo il led
13   delay ( tempo ); // aspetto un tempo
14   digitalWrite ( LED, LOW ); // spengo il led
15   delay ( tempo ); // aspetto un tempo
16
17 }
```

PROGRAMMAZIONE Le variabili:

Nel “ `setup()` ” corregiamo “ `pinMode` ” sostituendo il numero due con “LED” in questo modo:

```
pinMode ( LED, OUTPUT );
```

Ora abbiamo scritto il programma in un modo più corretto perché è più logico e se vogliamo modificare una connessione sarà sufficiente cambiare il numero del comando “ `#define` ”.

Per usare meglio la variabile “tempo” possiamo montare un potenziometro per regolare la frequenza del lampeggio e capire come funzionano e si usano gli ingressi analogici e il potenziometro.

IL CONVERTITORE ANALOGICO - DIGITALE:

Il convertitore analogico – digitale (ADC dall'inglese "Analog-to-Digital Converter") è un dispositivo elettronico capace di convertire un segnale analogico in un dato numerico binario.

Arduino Uno ha sei ingressi analogici collegati a un convertitore ADC con una risoluzione di 10 bit. A questi ingressi possiamo collegare tensioni da 0 a 5 Volt. Sono le uniche entrate che possono accettare segnali analogici.

Un segnale digitale è un segnale che ha solo due livelli lo zero (-V, GND) e l'uno (+V, VCC, la tensione operativa del micro controllore).

Un segnale analogico può avere tutti i livelli di tensione compresi tra zero e la tensione di lavoro del micro controllore. Se al convertitore analogico digitale diamo in ingresso una tensione superiore a quella operativa del micro controllore lo bruciamo.

Un convertitore da 10 bit può dividere la tensione massima di ingresso (nel nostro caso con Arduino Uno 5 Volt) in 1024 divisioni perché 1024 è il numero più grande che si può ottenere con un Byte di 10 bit.

Se dividiamo 5 Volt per 1024:

$$5 / 1024 = 0.0048828$$

otteniamo un valore molto piccolo.

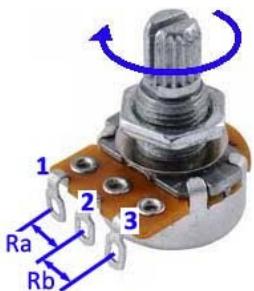
Il convertitore del microcontrollore di Arduino Uno ha una risoluzione di 5 mV (mili Volt).

IL CONVERTITORE ANALOGICO - DIGITALE:

Per fare una prova pratica usiamo un potenziometro come partitore di tensione.

I potenziometri, come le resistenze, hanno molti valori.

Prima di collegarlo dobbiamo conoscere quanta corrente consuma.



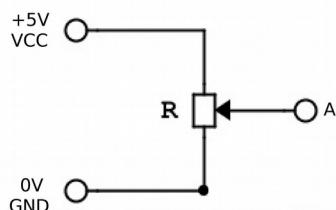
Il valore della resistenza di un potenziometro si misura tra le connessioni 1 e 3.

Per usarlo come partitore di tensione dobbiamo mettere a GND (0 Volt) la connessione 1 e a 5 Volt la connessione 3.

Usiamo la legge di Ohm per sapere quanta corrente consuma e se possiamo collegarlo ad Arduino.

Proviamo con un potenziometro da 4700 Ohm (4.7 KΩ).

Per sapere quanta corrente consuma la legge di Ohm è:



$$I = V/R$$

dove V è igual a 5 Volt y R è uguale a 4,700 Ohm.

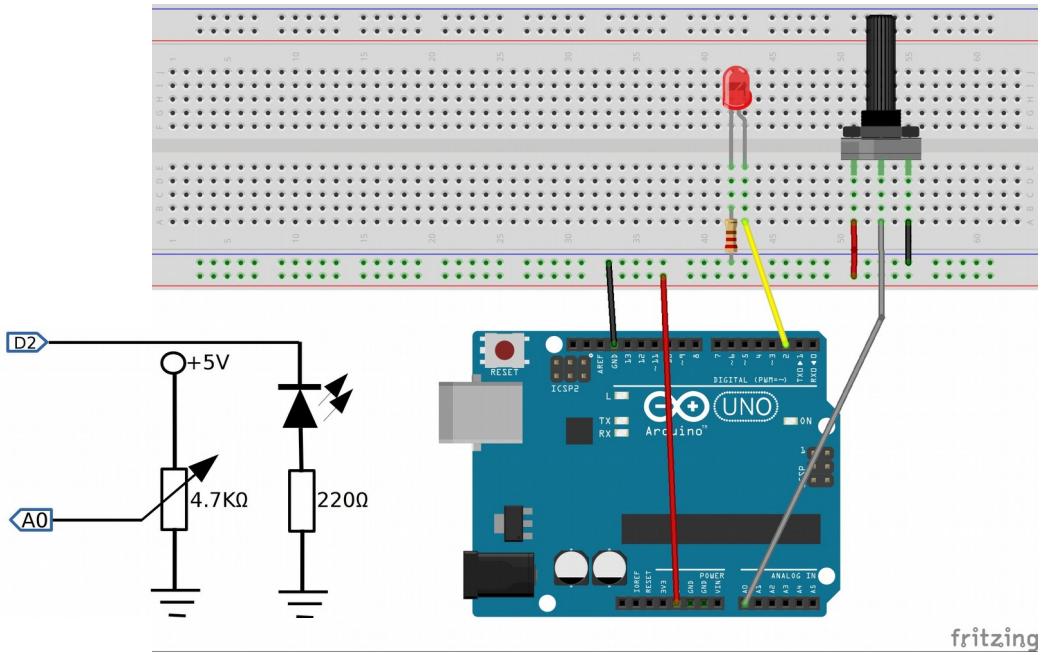
$$5/4,700 = 0.001$$

consuma appena 1mA (milli Amper) è perfetto.



La connessione 2 del potenziometro la collegiamo all'ingresso analogico "A0" di Arduino. Tutti gli ingressi analogici hanno la lettera "A" davanti al numero dell'ingresso.

IL CONVERTITORE ANALOGICO - DIGITALE:



In figura si può vedere come collegare il potenziometro.

Possiamo tornare al nostro programma.

PROGRAMMAZIONE - LEGGERE UN INGRESSO ANALOGICO:

Per leggere un ingresso analogico non serve scrivere nulla nel “setup()” e possiamo leggere il valore con l’istruzione:

analogRead (0)

dove “analogRead” significa “leggi l’ingresso analogico” e il numero nella parentesi rotonda è il numero dell’ingresso.

Quando scriviamo questa istruzione dobbiamo mettere il valore letto in una variabile per poterlo usare.

Noi dobbiamo leggere il valore dell’ingresso analogico per definire la frequenza del lampeggio del LED quindi mettiamo il valore letto nella variabile “tempo”.

tempo = **analogRead (0);**

Ora il valore letto sull’ingresso analogico (0) si memorizza nella variabile cancellando il valore precedente.

Dal momento che il convertitore analogico digitale legge un valore da 0 a 1023 per la nostra prova è perfetto perché prima avevamo messo un valore di 1000 nella variabile. Aggiungiamo la nuova istruzione come prima istruzione nel “loop()”.

```
1 #define LED 2 // è la connessione del LED
2
3 int tempo = 1000; // memorizza la frequenza del lampeggio
4
5 void setup() {
6   pinMode ( 2, OUTPUT ); // setto come uscita la connessione 2
7 }
8
9 void loop() {
10   tempo = analogRead(0); // leggo il valore di A0 e lo metto in "tempo"
11   digitalWrite ( 2, HIGH ); // accendo il led
12   delay ( tempo ); // aspetto un secondo
13   digitalWrite ( 2, LOW ); // spengo il led
14   delay ( tempo ); // aspetto un secondo
15 }
16 }
```

PROGRAMMAZIONE - LEGGERE UN INGRESSO ANALOGICO:

Correggiamo tutto il commentario perché ora il nostro programma ha un altro senso logico che è:

- leggo il valore di A0 e lo metto nella variabile "tempo"
- accendo il LED
- aspetto il valore della variabile "tempo"
- spengo il LED
- aspetto il valore della variabile "tempo"

carichiamo il programma in Arduino e, mettendo il potenziometro al massimo (girando l'albero in senso orario), il LED lampeggerà come nel primo Sketch perché il valore della tensione nell'ingresso analogico è di 5 Volt e il convertitore legge un valore di 1023 e il programma aspetta 1023 millisecondi.

Se abbassiamo il potenziometro poco a poco (girandolo in senso anti orario) possiamo vedere la frequenza del lampeggio aumentare.

Ad un certo punto vedremo il LED acceso fisso ma in realtà è solo perché sta lampeggiando più veloce di quello che il nostro occhio può distinguere.

Bene adesso colleghiamo il LED alla connessione digitale 3 di Arduino e vedremo il LED spegnersi perché non è programmata.

Nel nostro Sketch sostituiamo il numero 2 con il numero 3 al "#define" della connessione del LED in questo modo.

```
#define LED 3
```

Ora ricarichiamo lo Sketch in Arduino e il LED tornerà ad accendersi.

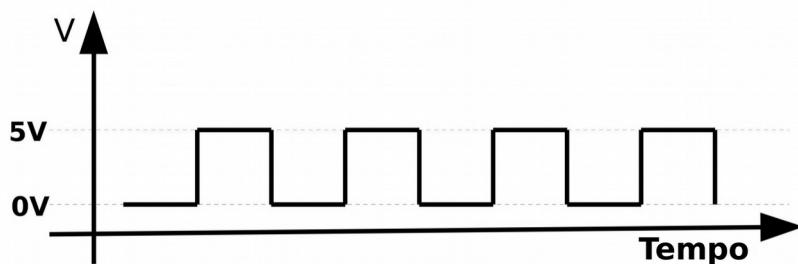
Alcune connessioni digitali di Arduino presentano al fianco del numero della connessione questo simbolo “ ~ ”, che ci dice che quella connessione ha anche la possibilità di avere un segnale di tipo PWM e si può usare come uscita analogica. Per l'esattezza non è un vero e proprio segnale analogico ma è una frequenza.

ARDUINO – USCITE ANALOGICHE - PWM

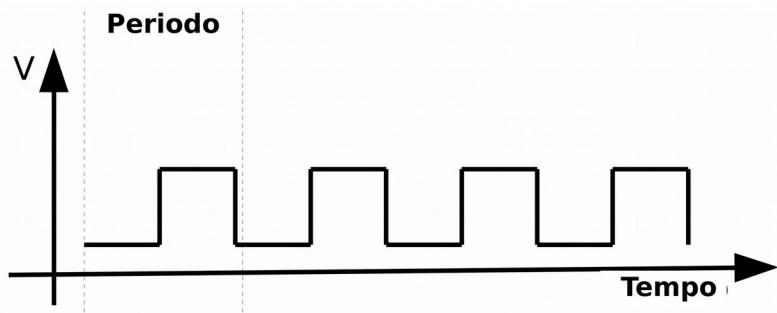
Il segnale PWM è un segnale che passa dallo stato logico 0 allo stato logico 1 molte volte al secondo perché ha una frequenza di 976 o 433 Hz (Hertz) il che significa che cambia di stato logico 976 o 433 volte al secondo.

Questo tipo di segnale si usa per regolare la velocità di un motore in corrente continua, variare la luce emessa da un LED o da un display LCD e per comandare un tipo particolare di motori che si chiamano servomotori e molte altre applicazioni.

Un segnale logico che passa da 0 a 1 crea una onda che si chiama Quadra.



Osservando la figura si può capire il perché di questo nome.

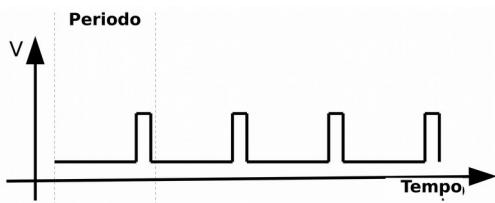


Si chiama “Periodo” una onda completa. Un’onda è completa quando ritorna al punto iniziale.

In figura possiamo vedere un’onda quadra perfetta perché la parte alta è identica a quella bassa.

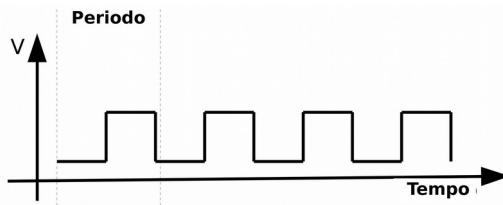
ARDUINO – USCITE ANALOGICHE - PWM

Il PWM non cambia la frequenza dell'onda quadra ma cambia la durata della semionda positiva (il tempo che l'onda sta alta). Il PWM accetta valori da 0 a 255. Se impostiamo un valore basso in uscita, per esempio 60, l'onda avrà un tempo di fronte alto molto stretto e un fronte basso molto ampio.



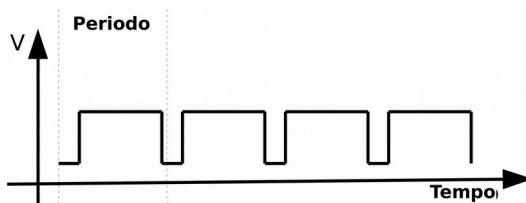
In questo modo in un secondo avremo sempre 976 o 433 onde complete (Periodi) ma sarà maggiore il tempo che avremo una uscita di livello basso.

Se impostiamo un valore medio, per esempio 128 (che è la metà di 255) avremo in uscita una onda quadra perfetta dove il fronte alto sarà uguale a quello basso.



In questo modo in un secondo il tempo che in uscita avremo un livello logico basso sarà uguale al tempo che avremo un livello logico alto.

Se impostiamo un valore alto, per esempio "200", l'onda avrà un fronte alto molto ampio e un fronte basso molto stretto.



In questo modo in un secondo il tempo che avremo in uscita un segnale alto sarà molto maggiore rispetto al tempo che avremo un segnale basso.

ARDUINO – USCITE ANALOGICHE - PWM

Se impostiamo a 0 il segnale PWM in uscita avremo un segnale basso e se lo impostiamo a 255 in uscita avremo un segnale alto come se stessimo scrivendo una uscita digitale.

Ora è chiaro perché con il segnale PWM si possa modificare la luce emessa da un LED, da un display o da una lampadina; con il PWM possiamo decidere quanto tenerli alimentati in un secondo. Il nostro occhio non percepisce il lampeggio ma percepisce un cambiamento di luminosità.

PROGRAMMAZIONE Le uscite analogiche:

Con il circuito che abbiamo montato sulla nostra basetta di prova proviamo a modificare la luminosità del LED agendo sul potenziometro.

Però abbiamo un piccolo problema: il convertitore analogico digitale restituisce valori da 0 a 1023 mentre il PWM può essere impostato da 0 a 255.

Per adattare la lettura analogica all'uscita PWM dobbiamo solo dividere per 4.

Dividere il valore di una variabile è molto semplice, si usa questa sintassi per tutte le operazioni matematiche.

nome della variabile = nome della variabile / 4;

Il micro controllore legge il valore della variabile, lo divide per 4 e lo sovrascrive al vecchio valore. Ora la variabile ha il risultato dell'operazione. Questo vale per tutti i tipi di operazione.

Per scrivere un nuovo Sketch per variare la luminosità del LED leggendo il valore del potenziometro il ragionamento logico è:

Come prima cosa leggo il valore del potenziometro altrimenti non ho nessun valore per impostare l'uscita analogica.

Dopo divido per 4 il valore letto perché il valore letto potrebbe essere da 0 a 1023 e l'uscita analogica chiede al massimo un valore di 255 e $1023 / 4 = 255$.

Scrivo il valore nell'uscita analogica.

Per scrivere un valore nell'uscita analogica la sintassi è:

analogWrite (uscita, valore);

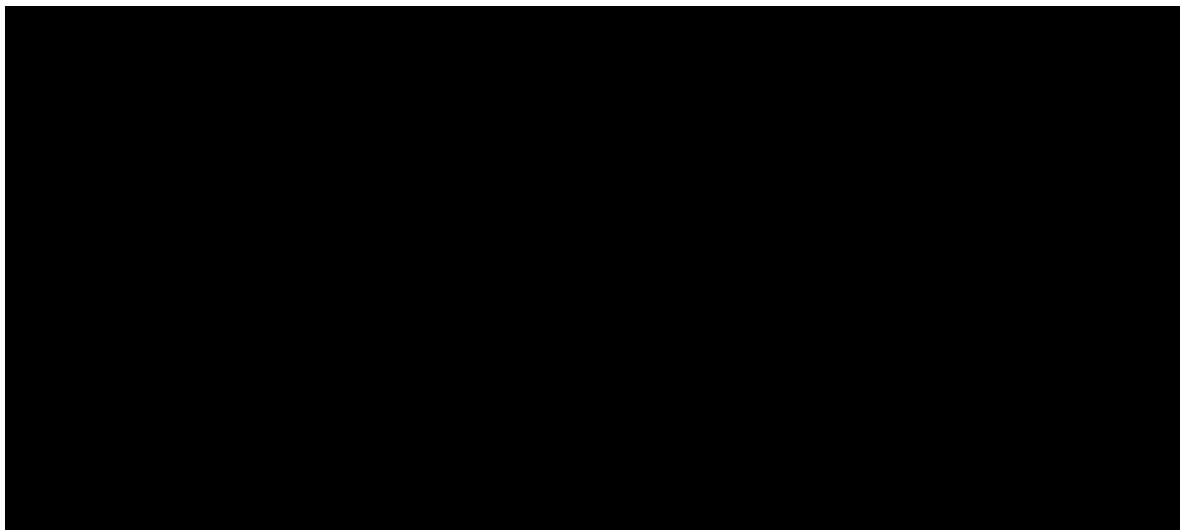
esattamente come scrivere una uscita digitale.

Se voglio scrivere il valore di PWM “100” nell'uscita analogica 3 posso scrivere:

analogWrite (3, 100);

PROGRAMACION Le uscite analogiche:

Scriviamo un nuovo Sketch usando i tasti “Ctrl” e “n” e salviamo come “potenziometro”.



In linea 1 creiamo la costante LED con il valore “3” che è dove abbiamo collegato il LED

in linea 2 creiamo la costante POTENZIOMETRO con il valore di “0” que è dove abbiamo collegato il potenziometro

in linea 4 creiamo una variabile di tipo “int” che chiamiamo “valore” per memorizzare la lettura del potenziometro

in linea 6, nel “setup()” impostiamo la connessione del LED (la numero 3) come uscita

in linea 12, nel “loop()”, mettiamo nella variabile “valore” la lettura del potenziometro

in linea 13 divido per 4 il valore della variabile per scrivere l'uscita analogica PWM

in linea 14 scivo il valore della variabile “valore” nell'uscita del LED

PROGRAMACION Le uscite analogiche:

Carichiamo il programma in Arduino.

Ora girando l'alberino del potenziometro il LED si accenderà più o meno luminoso. Con questo programma abbiamo capito come si possa cambiare la luminosità di un LED o di un display.

Sarebbe bello poter leggere il valore del potenziometro per capire meglio quello che succede nel programma.

PROGRAMMAZIONE Il monitor seriale:

Arduino può inviare dati che si possono vedere sul monitor del computer tramite la porta USB.

Quando usiamo la comunicazione seriale verso il computer o altre schede connesse ad Arduino non possiamo usare le connessioni 0 e 1 perché si trasformano in TX e RX, infatti se osserviamo la scheda possiamo vedere che a lato del numero delle due connessioni troviamo le scritte TX e RX (invio e ricezione dati).

Per inviare un dato al computer dobbiamo aprire la porta di comunicazione seriale in questo modo.

Serial.begin (9600);

Serial.begin apre la porta di comunicazione e il numero 9600 tra parentesi indica la velocità di trasmissione dei dati.

La quantità di dati trasmessi in un secondo si misura in Baud che indica bit per secondo quindi 9600 vuol dire che in un secondo inviamo 9600 bit.

Per inviare un dato usiamo questa sintassi.

Serial.print ();

in questo modo scriviamo un dato nella porta seriale che possiamo vedere sul monitor.

Se nella parentesi scriviamo il nome di una variabile sul monitor si scrive il valore della variabile.

Per esempio;

Serial.print (valore);

Se vogliamo scrivere una o più parole le dobbiamo racchiudere tra apici.

Serial.print ("Buon giorno");

In questo modo scriviamo **Buon giorno** sul monitor.

PROGRAMMAZIONE Il monitor seriale:

Con l'istruzione **Serial.print** i dati si scrivono uno dietro l'altro, se vogliamo portare a capo il cursore dobbiamo aggiungere il suffisso **ln**.

Serial.println (valore);

Salviamo il programma prima di modificarlo come “seriale”.
Mettiamo in pratica la nuova istruzione:

```
1 #define LED 3 // è la connessione del led
2 #define POTENZIOMETRO 0 // è la connessione del potenziometro
3
4 int valore = 0; // in "valore" memorizzo la lettura del potenziometro
5
6 void setup() {
7   pinMode ( LED, OUTPUT ); // configuro la connessione LED come uscita
8   Serial.begin ( 9600 ); // attivo la porta di comunicazione
9 }
10
11 void loop() {
12   valore = analogRead ( POTENZIOMETRO ); // in "valore" metto la lettura del potenziometro
13   Serial.println ( valore ); // scrivo il valore sul monitor
14   delay ( 500 ); // aspetto mezzo secondo
15   valore = valore / 4; // divido la variabile
16   analogWrite ( LED, valore ); // scrivo sull'uscita analogica del LED il valore della variabile
17
18 }
```

in linea 8: come prima cosa scriviamo nella funzione “setup()” l'istruzione per aprire la porta di comunicazione seriale.

Per vedere il valore del potenziometro l'istruzione che scrive il dato sulla porta seriale la dobbiamo inserire sotto l'istruzione che legge il valore del potenziometro prima che venga divisa per 4.

in linea 13: qui scriviamo l'istruzione “**Serial.println (valore);**” con il suo commento.

In linea 14: qui mettiamo un piccolo tempo di delay di mezzo secondo per avere il tempo di leggere i dati sul monitor. Arduino è molto veloce a scrivere i dati se non lo rallentiamo non avremo tempo per leggere.

PROGRAMMAZIONE Il monitor seriale:

Carichiamo il programma in Arduino e apriamo il monitor seriale.

Per aprire il monitor seriale facciamo clic sopra la lente.



In basso a destra nella finestra del monitor seriale dobbiamo scrivere la velocità dei dati che deve essere uguale a quella che abbiamo dichiarato nelle parentesi del "Serial.begin".

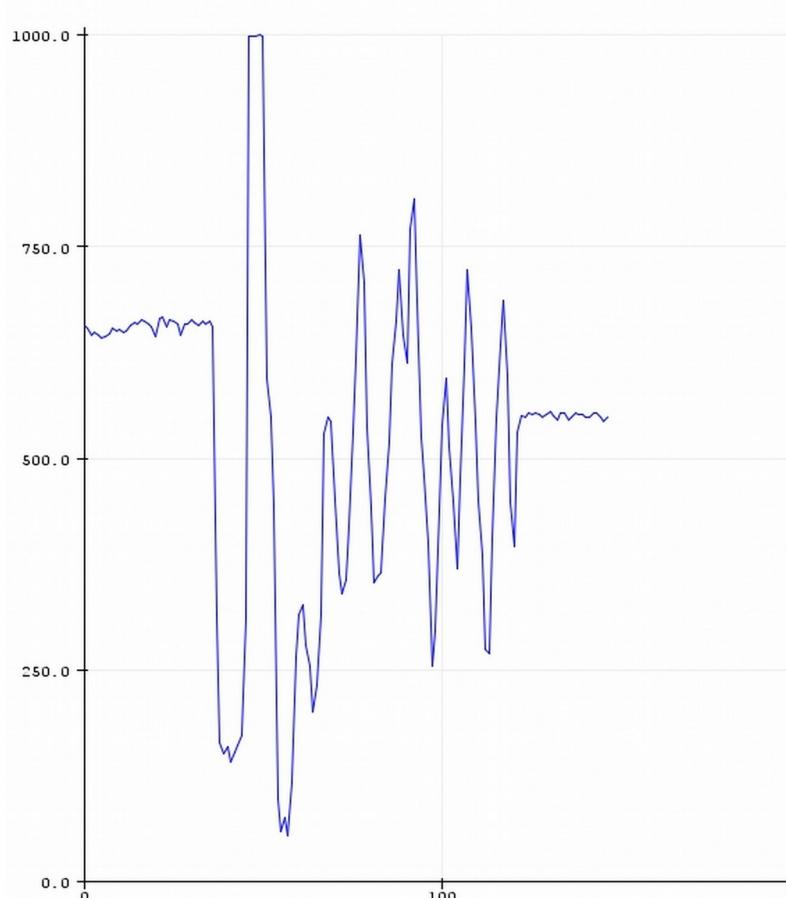


Girando l'alberino del potenziometro possiamo vedere sul monitor il valore che Arduino legge sull'ingresso analogico.



PROGRAMMAZIONE Il monitor seriale:

L'IDE possiede anche un plotter seriale che troviamo nella barra dei menu "Strumenti".



Il serial plotter crea un disegno grafico dei valori. Girando l'alberino del potenziometro si possono vedere i valori nel grafico.

Ora possiamo mettere un valore molto basso nella funzione delay come un 30 e caricare il programma per rendere il grafico molto reattivo.

PROGRAMMAZIONE Il monitor seriale:

Aggiungendo una linea di codice nello Schetch possiamo leggere il valore in Volt in uscita dal partitore di tensione (il potenziometro) per vedere in pratica come funziona.

```
1 #define LED 3 // è la connessione del led
2 #define POTENZIOMETRO 0 // è la connessione del potenziometro
3
4 int valore = 0; // in "valore" memorizzo la lettura del potenziometro
5 float VOLT = 0; // in volt memorizzo la tensione che ho nell'ingresso analogico
6 void setup() {
7   pinMode ( LED, OUTPUT ); // configuro la connessione LED come uscita
8   Serial.begin ( 9600 ); // attivo la porta di comunicazione
9 }
10
11 void loop() {
12   valore = analogRead ( POTENZIOMETRO ); // in "valore" metto la lettura del potenziometro
13   VOLT = valore * 0.0048; // moltiplico per il valore della risoluzione del convertitore
14   Serial.println ( VOLT ); // scrivo i volt sul monitor
15   delay ( 500 ); // aspetto mezzo secondo
16   valore = valore / 4; // divido la variabile
17   analogWrite ( LED, valore ); // scrivo sull'uscita analogica del LED il valore della variabile
18
19 }
```

Creiamo una variabile di tipo **float** per immagazzinare valori numerici con la virgola che chiamiamo VOLT dove memorizzeremo il valore in volt all'ingresso analogico.

Sotto la prima linea del “loop()” (in linea 13) aggiungiamo:

VOLT = valore * 0.0048 ;

0.0048 è il valore di tensione più piccolo che può leggere il convertitore .

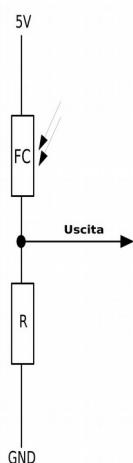
Carichiamo il programma e sul monitor seriale possiamo leggere il valore in Volt che esce dal potenziometro.

ARDUINO COLLEGARE UNA FOTORESISTENZA

Nel nostro smartphone o nel tablet c'è un programma che misura la luce ambientale e regola in automatico la luminosità dello schermo perché sarebbe scomodo avere uno schermo molto luminosa la notte e non vedere nulla di giorno.

Se collegiamo una fotoresistenza all'ingresso analogico di Arduino dove ora abbiamo collegato il potenziometro possiamo regolare la luminosità del LED con la luce del giorno.

Per misurare il valore della resistenza dobbiamo sempre fare un partitore di tensione.



Per conoscere il valore di tensione in uscita prendiamo la formula del partitore di tensione.

$$V_{\text{uscita}} = V_{\text{ingresso}} * \frac{R}{FC + R}$$

La fotoresistenza ha un valore che dipende dalla luce che vede, nell'oscurità totale è di $1M\Omega$ (1 Mega Ohm 1000000 di Ohm) e quando vede molta luce scende a 10Ω (10 Ohm) o meno.

Nella formula “V entrata” è uguale a 5 Volt, come R mettiamo una resistenza da 100 K Ω (100 Kilo Ohm = 100000 Ohm) e come valore della fotoresistenza nel buio mettiamo 1 M Ω (1000000 di Ohm).

$$5 \text{ V} * \frac{100,000}{1,000,000 + 100,000} = 0.45 \text{ V}$$

Senza luce in uscita dal partitore di tensione abbiamo una tensione molto bassa.

ARDUINO COLLEGARE UNA FOTORESISTENZA

Se ora spostiamo la fotoresistenza alla luce del sole il suo valore si abbassa a $10\ \Omega$ o meno. Proviamo a mettere nella formula il valore 10.

$$5\text{ V} * \frac{100,000}{10 + 100,000} = 4.99\text{ V}$$

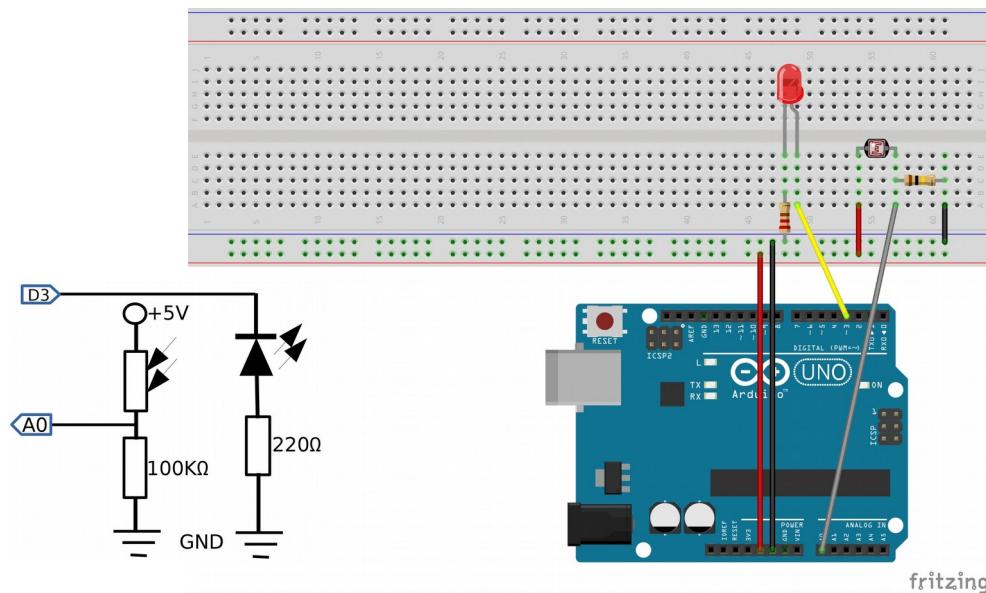
Se abbiamo molta luce in uscita dal partitore di tensione il voltaggio è molto alto.

Con un altro valore di luce la fotoresistenza potrebbe avere un valore di $75\text{ K}\Omega$ (75000 Ohm).

$$5\text{ V} * \frac{100,000}{75,000 + 100,000} = 2.85\text{ V}$$

È chiaro come il voltaggio cambia con la luce che colpisce la fotoresistenza.

Ora montiamo la fotoresistenza e la resistenza sulla basetta di prova e colleghiamola come sullo schema di cablaggio.



ARDUINO COLLEGARE UNA FOTORESISTENZA

Apriamo il programma che abbiamo salvato con il nome “potenziometro” e carichiamolo in Arduino.

Ora il LED avrà molta luce se la fotoresistenza riceverà molta luce e se mettiamo una mano sopra la fotoresistenza per fare ombra la luminosità del LED si abbasserà.

Se copriamo con la mano la fotoresistenza in modo che non veda più luce il LED si spegnerà.

Però lo schermo dello smartphone o del tablet non si spengono la notte perché il software mantiene un valore minimo di luminosità per permetterci di vederlo.

Possiamo fare la stessa cosa.

PROGRAMMAZIONE - Il controllo di flusso:

Il controllo di flusso permette ad Arduino di prendere delle decisioni e questo lo rende 'intelligente'.

Possiamo dire ad Arduino: " Se succede questo devi fare questa cosa se no vai a fare questa altra cosa."

Arduino può decidere se un valore è:

- Maggiore >
- Minore <
- Uguale ==
- Diverso !=
- Minore o uguale >=
- Maggiore o uguale <=

Possiamo dire ad Arduino:

" Se il valore della variabile è uguale a 300 spegni il LED se no accendi il LED."

Per scrivere questa istruzione si usa il controllo di flusso "if" che in inglese vuol dire "se".

```
If ( variabile == 300){  
    digitalWrite ( LED, HIGH);  
}
```

Nella parentesi rotonda scriviamo la condizione e nelle parentesi graffe le istruzioni che deve eseguire se la condizione è vera.

```
else {  
    digitalWrite ( LED, LOW );  
}
```

"else" vuol dire "se no", quindi se non è vera la condizione si eseguono le istruzioni nelle parentesi graffe del "else".

Non è obbligatorio mettere l'"else", se non lo mettiamo dopo il "if" il programma segue il suo corso.

PROGRAMMAZIONE - Il controllo di flusso:

Quindi per impedire al LED di scendere sotto una certa luminosità, inseriamo un controllo di flusso nel nostro programma.

Salviamo il programma prima di modificarlo e chiamiamolo “fotoresistenza”.

```
1 #define LED 3
2 #define POTENZIOMETRO 0
3
4 int valore = 0; // in valore metto la lettura del potenziometro
5
6 void setup() {
7     pinMode ( LED, OUTPUT ); // configuro la connessione LED come uscita
8
9 }
10
11 void loop() {
12     valore = analogRead ( POTENZIOMETRO ); // in "valore" metto la lettura del potenziometro
13     valore = valore / 4; // divido la variabile
14     analogWrite ( LED, valore ); // scrivo sull'uscita analogica del LED il valore della variabile
15
16 }
```

Ora dobbiamo mettere un limite alla luminosità del LED in modo che non si spenga nel buio.

Per dirlo in modo logico possiamo pensare che se il valore del PWM si abbassa sotto il 100, quindi è minore di 100, lo rimettiamo a 100.

Prima di scrivere l'uscita analogica inseriamo il controllo di flusso, una condizione.

Abbassiamo la linea 14 e scriviamo in linea 14:

```
if ( valore < 100 ){
```

Se dopo aver aperto la parentesi graffa pigiamo il tasto “enter” sulla tastiera l'IDE libera una linea e chiude automaticamente la parentesi.

PROGRAMMAZIONE - Il controllo di flusso:

Nelle parentesi graffe scriviamo:

```
valore = 100;
```

in questo modo il valore non può scendere sotto il 100 perché se si abbassa lo riscriviamo a 100 cancellando il valore precedente.

```
1 #define LED 3
2 #define FOTORESISTENZA 0
3
4 int valore = 0; // in valore metto la lettura della fotoresistenza
5 void setup() {
6     pinMode ( LED, OUTPUT ); // configuro la connessione LED come uscita
7
8 }
9
10 void loop() {
11     valore = analogRead ( FOTORESISTENZA ); // in "valore" metto la lettura della fotoresistenza
12     valore = valore / 4; // divido la variabile
13     if ( valore < 100 ) { // se il valore della variabile "valore" è minore di 100
14         valore = 100; // metto a 100 la variabile
15     }
16     analogWrite ( LED, valore ); // scrivo sull'uscita analogica del LED il valore della variabile
17
18 }
```

Carichiamo il programma in Arduino. Ora se mettiamo la mano sulla fotoresistenza la luminosità del LED si abbasserà fino al valore di PWM 100 e si fermerà.

È così che funziona il sistema di regolazione automatica della luminosità del display dello smartphone o del tablet.

PROGRAMMAZIONE - Il controllo di flusso:

Scopriamo come funziona una fotoresistenza che accende una lampada di notte e la spegne di giorno. La fotoresistenza accende la lampada quando la luce del giorno si abbassa sotto un certo livello e la spegne quando la luce supera quel livello.

Apriamo un nuovo Sketch premendo i tasti “Ctrl e “n” sulla tastiera e salviamo come “fotoresistenza 2”:

Ora dobbiamo:

- leggere il valore della fotoresistenza.
- se il valore è minore di 500 accendiamo il LED.
- se no, spegnamo il LED.

Escribimos el programa:

```
1 #define LED 3
2 #define FOTORESISTENZA 0
3
4 int valore = 0; // in valore metto la lettura della fotoresistenza
5 void setup() {
6   pinMode ( LED, OUTPUT ); // configuro la connessione LED come uscita
7
8 }
9
10 void loop() {
11   valore = analogRead ( FOTORESISTENZA ); // in "valore" metto la lettura della fotoresistenza
12
13   if ( valore < 500 ) { // se il valore della variabile "valore" è minore di 500
14     digitalWrite ( LED, HIGH ); // accendo il LED
15   }
16   else { // se no
17     digitalWrite ( LED, LOW ); // spengo il LED
18   }
19 }
20 }
```

In linea 1 e 2 scriviamo le costanti e nella linea 3 creiamo la variabile di tipo “int” che chiamiamo “valore”

Nel “setup()” impostiamo la connessione LED come uscita

nella linea 11 leggiamo il valore dell'entrata analogica

in linea 13 mettiamo la condizione (un controllo di flusso): se il valore è minore di 500 (il valore 500 è un valore basso di luce ma si può provare con altri valori)

in linea 14 (dentro le parentesi graffe del “if”) impostiamo alta l'uscita del LED

in linea 16 “se non è minore di”

in linea 17 nelle parentesi graffe dell' “else” portiamo bassa l'uscita del LED per spegnerlo.

Carchiamo il programma in Arduino e se mettiamo la mano sopra la fotoresistenza il LED si accenderà e se la togliamo si spegnerà.

PROGRAMMAZIONE - Il controllo di flusso:

Esiste un altro controllo di flusso molto utile è il “for”.

Il “for” può eseguire le istruzioni racchiuse nelle sue parentesi graffe un certo numero di volte. È molto comodo se ad un certo punto del nostro programma ci serve eseguire un gruppo di istruzioni sempre uguali per le volte che vogliamo.

La sintassi è:

```
for ( int i=0; i<10; i++){  
    // istruzioni  
}
```

Nella parentesi rotonda scriviamo le tre istruzioni necessarie:

- la prima è la creazione di una variabile che mettiamo a zero,
- la seconda comunica al “for” che deve eseguire le istruzioni nelle parentesi graffe finché la variabile è minore di 10,
- la terza è “i++”.

Scrivere “nome variabile ++” è come scrivere
“ nome variabile = nome variabile + 1”

quindi “ i++ ” è come scrivere “ i = i + 1 ”

il microcontrollore prende il valore della variabile, somma 1 e la rimemorizza nella variabile cancellando il valore precedente.

Se avevamo un valore di 5 dopo avremo un valore di 6.

in questo modo tutte le volte che esegue le istruzioni nelle sue parentesi graffe somma 1 alla variabile. Quando la variabile avrà un valore di 10 non sarà più minore di 10 e il “for” si fermerà e il programma potrà proseguire.

In questo esempio il “for” ripete per 10 volte le istruzioni nelle sue parentesi graffe.

Possiamo mettere in pratica questa nuova istruzione di controllo.

PROGRAMMAZIONE - Il semaforo:

Scopriamo come funziona un semaforo americano.

In un incrocio abbiamo due semafori uno per una strada e uno per l'altra.

La prima cosa che dobbiamo sapere è come si presentano i semafori quando si accendono per la prima volta o dopo che si riavviano perché è mancata la corrente elettrica.

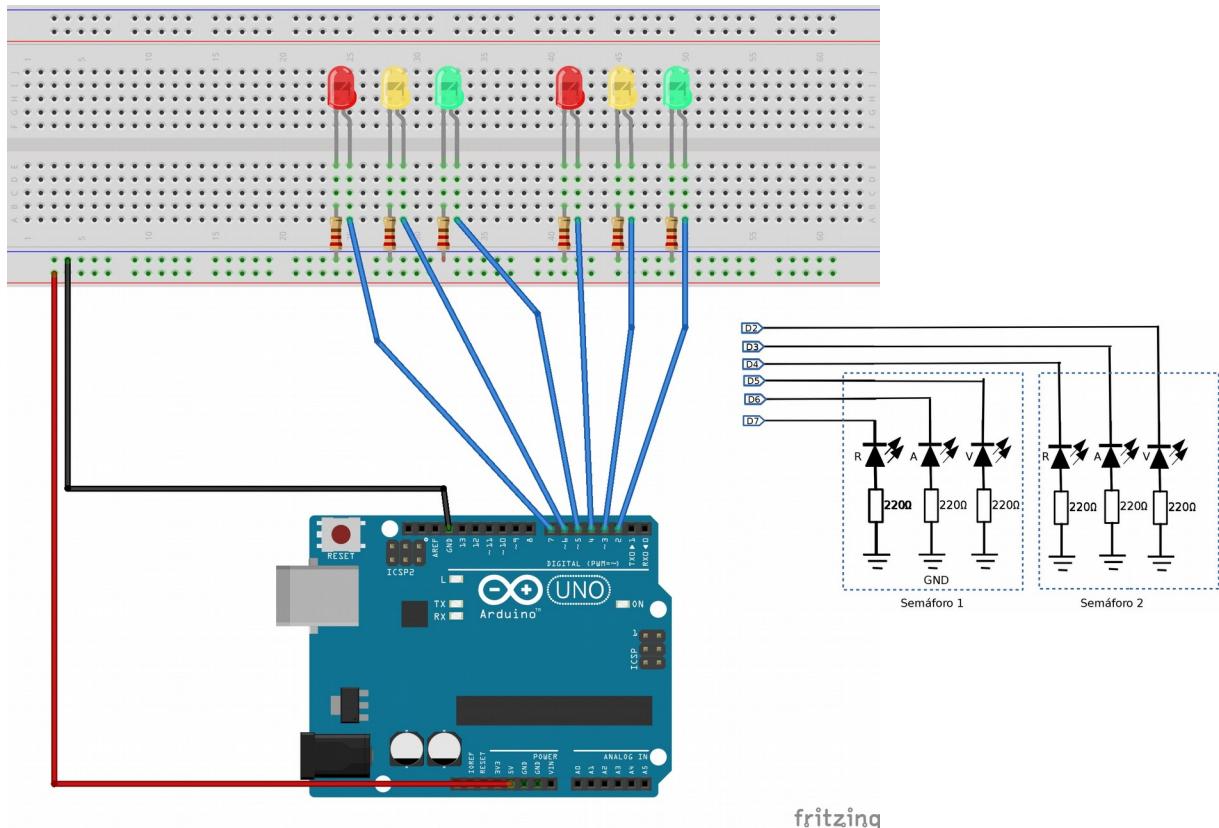
Per logica deve accendersi con il rosso per entrambe le strade in modo da fermare il traffico prima di mettersi in funzione.

Scriviamo il ragionamento logico prima di scrivere il programma in modo d'avere le idee chiare.

- Accendo il rosso per tutti.
- Aspetto 5 secondi per fermare il traffico.
- Spengo il rosso del semaforo 1 e accendo il verde al semaforo 1
- Aspetto 15 secondi
- Faccio lampeggiare il verde quattro volte al semaforo 1
- Spengo il verde al semaforo 1 e accendo il giallo
- Aspetto due secondi
- Spengo il giallo e accendo il rosso al semaforo 1
- Aspetto 5 secondi che si ferma il traffico
- Spengo il rosso e accendo il verde del semaforo 2
- Aspetto 15 secondi
- Faccio lampeggiare 4 volte il verde del semaforo 2
- Spengo il verde e accendo il giallo del semaforo 2
- Aspetto 2 secondi
- Spengo il giallo del semaforo 2

Per fare due semafori ci servono 2 LED rossi, 2 LED gialli, 2 LED verdi e 6 resistenze da 220Ω che montiamo sulla basetta di prova come nello schema di montaggio.

PROGRAMMAZIONE - Il semaforo:



Ora che abbiamo montato tutti i componenti sulla basetta di prova e abbiamo collegato Arduino possiamo scrivere lo Sketch.

Come prima cosa scriviamo la parte alta del programma con le costanti.

```
1 #define LED_verde_1 2      // collego il LED verde del semaforo 1 a la connessione 2
2 #define LED_giallo_1 3     // collego il LED giallo del semaforo 1 a la connessione 3
3 #define LED_rosso_1 4      // collego il LED rosso del semaforo 1 a la connessione 4
4 #define LED_verde_2 5      // collego il LED verde del semaforo 2 a la connessione 5
5 #define LED_giallo_2 6     // collego il LED giallo del semaforo 2 a la connessione 6
6 #define LED_rosso_2 7      // collego il LED rosso del semaforo 2 a la connessione 7
```

PROGRAMMAZIONE - Il semaforo:

Poi nel “setup()” impostiamo le connessioni:

```
7
8 void setup() {
9   pinMode ( LED_verde_1, OUTPUT );    // imposto come uscita la connessione del LED_verde_1
10  pinMode ( LED_giallo_1, OUTPUT ); // imposto come uscita la connessione del LED_giallo_1
11  pinMode ( LED_rosso_1, OUTPUT );    // imposto come uscita la connessione del LED_rosso_1
12  pinMode ( LED_verde_2, OUTPUT );    // imposto come uscita la connessione del LED_verde_2
13  pinMode ( LED_giallo_2, OUTPUT ); // imposto come uscita la connessione del LED_giallo_2
14  pinMode ( LED_rosso_2, OUTPUT );    // imposto come uscita la connessione del LED_rosso_2
15
16 }
17
```

Ora possiamo scrivere la prima parte del programma.

```
17
18 void loop() {
19   digitalWrite( LED_rosso_1,HIGH); // accendo il rosso del semaforo 1
20   digitalWrite( LED_rosso_2,HIGH); // accendo il rosso del semaforo 2
21   delay(5000);                  // aspetto 5 secondi
22   digitalWrite( LED_rosso_1,LOW); // spegno il rosso del semaforo 1
23   digitalWrite( LED_verde_1,HIGH); // accendo il verde del semaforo 1
24   delay(15000);                // aspetto 15 secondi
```

Come prima cosa dobbiamo mettere il semaforo rosso per tutti e lo scriviamo in linea 19 e 20.

In linea 21 aspettiamo 5 secondi.

In linea 22 spegnamo il rosso del semaforo 1 e

in linea 23 accendiamo il verde al semaforo 1.

In linea 24 aspettiamo 15 secondi.

Ora dobbiamo far lampeggiare il verde 4 volte e lo facciamo usando il controllo di flusso “for”.

PROGRAMMAZIONE - Il semaforo:

```
25 | for (int i=0; i<4; i++){           // spengo e accendo il verde del semaforo 1 quattro volte
26 |   digitalWrite(LED_verde_1,LOW);
27 |   delay(250);
28 |   digitalWrite(LED_verde_1,HIGH);
29 |   delay (250);
30 | }
```

In linea 25 scriviamo l'istruzione "for" e nelle sue parentesi rotonde creiamo una variabile di tipo "int" che chiamiamo "i" e che mettiamo uguale a zero.

Questa variabile è di tipo locale e si può usare solo nel controllo di flusso "for".

Quando scriviamo: for (int i=0; i<4; i++){

Stiamo scrivendo questo:

per "i" da zero finché è minore di 4, esegui le tue istruzioni e poi somma 1 alla variabile "i".

Il ciclo si ripete 4 volte: quando "i" è uguale a zero, a uno, a due, a tre e quando "i" è uguale a quattro, non è minore di quattro e il "for" si fermerà.

Dalla linea 26 alla linea 29 nelle parentesi graffe del "for" scriviamo, come già sappiamo fare, le istruzioni per fare un lampeggio del LED con un tempo di 250 mS.

Abbiamo scritto un solo lampeggio ma ne abbiamo ottenuti 4.

La seconda parte del programma è identica alla prima, cambia solo il numero del semaforo.

Finiamo di scrivere il programma.

PROGRAMMAZIONE - Il semaforo:

```
31  digitalWrite( LED_verde_1,LOW); // spengo il verde del semaforo 1
32  digitalWrite( LED_giallo_1,HIGH); // accendo il giallo del semaforo 1
33  delay(2000); // aspetto 2 secondi
34  digitalWrite( LED_giallo_1,LOW); // spengo il giallo del semaforo 1
35  digitalWrite( LED_rosso_1,HIGH); // accendo il rosso del semaforo 1
36  delay (5000); // aspetto 5 secondo
37  digitalWrite( LED_rosso_2,LOW); // spengo il rosso del semaforo 2
38  digitalWrite( LED_verde_2,HIGH); // accendo il verde del semaforo 2
39  delay(15000); // aspetto 15 secondo
40  for (int i=0; i<4; i++){ // spengo e accendo il verde del semaforo 2 quattro volte
41    digitalWrite(LED_verde_2,LOW);
42    delay(250);
43    digitalWrite(LED_verde_2,HIGH);
44    delay (250);
45  }
46  digitalWrite( LED_verde_2,LOW); // spengo il verde del semaforo 2
47  digitalWrite( LED_giallo_2,HIGH); // accendo il giallo del semaforo 2
48  delay(2000); // aspetto 2 secondi
49  digitalWrite( LED_giallo_2,LOW); // spengo il led giallo del semaforo 2
50
51 }
```

Possiamo caricare il programma in Arduino e vedere come funziona un semaforo.

Il programma di un semaforo è semplice perché è solo una sequenza di istruzioni che non devono decidere nulla.

Tutte le macchine ricevono comandi dal mondo esterno, andiamo a scoprire come si collegano i pulsanti ad Arduino.

ARDUINO - COLLEGARE UN PULSANTE:

*pulsante
simbolo elettrico*



pulsante NO



pulsante NC

Come si può vedere in figura un pulsante può essere di tipo “Normalmente Chiuso” NC o “Normalmente Aperto” NA o NO.

Questo significa che se abbiamo un pulsante NA (Normalmente Aperto) e non lo premiamo la corrente non passa, passa solo se lo premiamo perché chiude il contatto.

Tuttavia se abbiamo un pulsante NC (Normalmente Chiuso) la corrente passa sempre e si interrompe se premiamo il pulsante perché il suo contatto si apre.

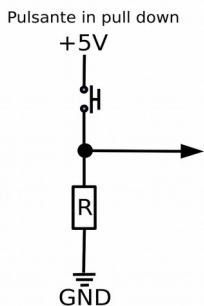
I pulsanti possono avere forme differenti



Come scrissi, Arduino è una scheda logica e i suoi ingressi possono essere collegati solo ad un livello di tensione di 5 V per il livello logico 1 (alto) o 0 V per il livello logico 0 (basso).

Se colleghiamo un pulsante NA (Normalmente Aperto) a un ingresso digitale di Arduino, se non lo premiamo è come se l'ingresso non fosse collegato a nulla e il microcontrollore impazzisce.

ARDUINO - COLLEGARE UN PULSANTE:

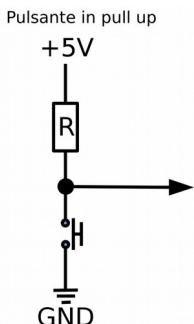


Per collegare un pulsante a Arduino dobbiamo mettere una resistenza di “Pull down” che vuol dire di spinta in basso. Lo schema elettrico è simile a quello del partitore di tensione che al posto di una resistenza ha il pulsante. In effetti un pulsante potrebbe essere una resistenza con due valori: uno molto alto quando il pulsante non è premuto (aperto) e l'altro a $0\ \Omega$ quando è premuto (chiuso).

In questo caso quando il pulsante è aperto l'ingresso è a livello logico basso (0 V) perché la corrente negativa passa dalla resistenza R che è minore della resistenza del pulsante che tende all'infinito.

Quando premiamo il pulsante la resistenza del pulsante è minore di quella della resistenza R e la corrente positiva raggiunge l'ingresso digitale portandolo a stato logico 1 (5 V).

La corrente elettrica, come l'acqua, è pigra e passa sempre dove fa meno fatica.



Se mettiamo una resistenza di “pull up” che vuol dire spinta in alto otterremo un funzionamento opposto. Quando il pulsante non è premuto (aperto) dalla resistenza R può passare la corrente positiva e sull'ingresso avremo un livello logico alto. Quando il pulsante è premuto (chiuso) la corrente negativa passerà per il pulsante con una resistenza minore ($0\ \Omega$) portando l'ingresso ad un livello logico basso.

Il microcontrollore di Arduino ha la possibilità di inserire automaticamente una resistenza di pull up sugli ingressi digitali in modo di collegare direttamente un pulsante però quando premeremo il pulsante il livello che leggeremo sarà basso e normalmente sarà alto.

Per il nostro cervello è più facile immaginare che un livello logico alto sia abbinato ad una azione e basso a nessuna azione. Lavorare con una resistenza di pull up ci obbliga a ragionare al contrario e questo è piuttosto scomodo per chi inizia a programmare.

PROGRAMMAZIONE-LEGGERE UN INGRESSO DIGITALE

Per leggere un ingresso digitale dobbiamo impostare la connessione come ingresso nella funzione “`setup()`” in questo modo:

`pinMode (connessione, INPUT);`

dove `PinMode` è l'istruzione che configura la connessione.

Nella parentesi rotonda si mette il numero della connessione che vogliamo configurare e dopo la virgola il modo `INPUT`

per configurare l'ingresso con una resistenza di pull up scriviamo:

`pinMode (connessione, INPUT_PULLUP);`

in questo modo il microcontrollore mette in automatico una resistenza di pull up.

Per leggere un ingresso digitale la sintassi è:

`digitalRead (ingresso);`

Mettiamo in pratica la nuova istruzione:

per farlo ci serve un pulsante, un LED, una resistenza da 10 KΩ (10 Kilo Ohm, 10000 Ohm) e una resistenza da 220 Ω.

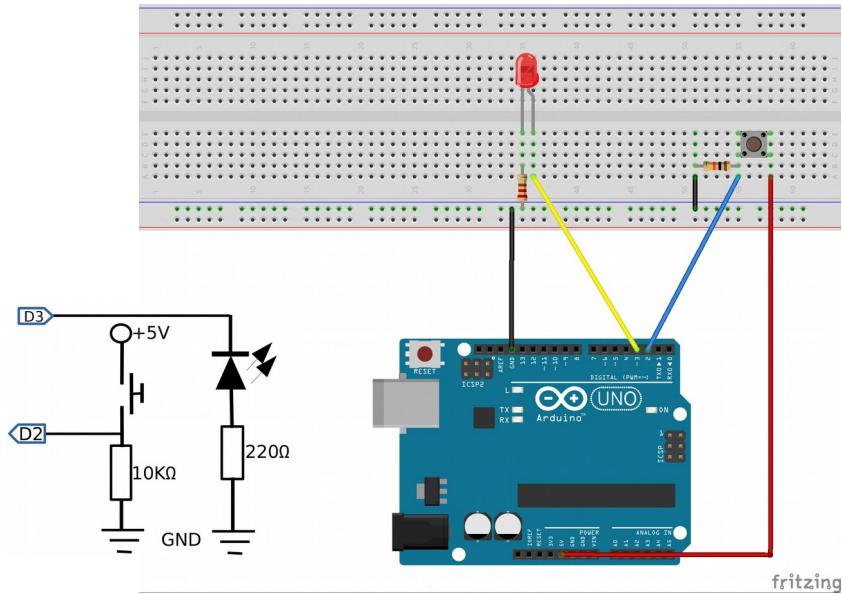
Per riconoscere la resistenza da 10 KΩ facciamo riferimento alla tavola dei colori.
Se abbiamo una resistenza con 4 anelli i colori saranno:

Marrone per il primo numero che è un 1 / nero per il secondo numero che è 0 / arancione per il numero di zeri che completano il numero che sono 3.

se la resistenza ha 5 anelli i colori saranno:

marrone per il primo numero che è 0 / nero per il secondo numero che è 0 / nero per il terzo numero che è 0 / e rosso perché restano due zeri per completare il numero.

PROGRAMMAZIONE-LEGGERE UN INGRESSO DIGITALE



Colleghiamo seguendo lo schema di montaggio della figura.

Dopodichè scriviamo un semplice Sketch per provare il pulsante.

Con questo programma se premiamo il pulsante il Led si accende se no si spegne.

```
1 #define PULSANTE 2
2 #define LED 3
3
4 void setup() {
5   pinMode ( PULSANTE, INPUT ); // configuro la connessione PULSANTE come ingresso
6   pinMode ( LED, OUTPUT ); // configuro la connessione LED come uscita
7 }
8
9 void loop() {
10  if ( digitalRead == HIGH ) { // se l'ingresso del pulsante è alto
11    digitalWrite ( LED, HIGH ); // accendo il LED
12  }
13  else { // se no
14    digitalWrite ( LED, LOW ); // spengo il LED
15  }
16 }
```

PROGRAMMAZIONE-LEGGERE UN INGRESSO DIGITALE

Scriviamo le costanti in alto alla linea 1 e 2.

Nel setup impostiamo la connessione del pulsante come ingresso e la connessione del LED come uscita, linea 5 e 6.

nel loop in linea 10 mettiamo un controllo di flusso “if” che controlla se l’ingresso del pulsante è alto. Se è così accende il LED.

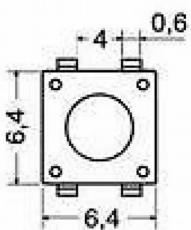
Se non lo è (else) lo spegne, linea 13.

Ricordatevi sempre che per dire uguale in un controllo di flusso si scrive “==”, se mettiamo un solo uguale l’IDE non segnala errore e il programma non funziona o funziona male.

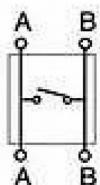
Salviamo il programma come “pulsante” e carichiamolo nella memoria di Arduino.

Ora premendo il pulsante il LED si accende e rilasciandolo si spegne.

Se il LED rimane sempre acceso occorre girare il pulsante di 90 gradi perché è quadrato e non si vede bene il senso giusto.



Questo tipo di pulsante ha due collegamenti “A” e due collegamenti “B”, se non lo si mette nel verso giusto rimane sempre chiuso.



Dobbiamo posizionarlo come si vede in figura.

PROGRAMMAZIONE - GIOCHIAMO :

A tutti, me compreso, piace giocare.

Abbiamo imparato molte cose del mondo di Arduino, dell'elettronica e della programmazione e ora che sappiamo collegare anche un pulsante possiamo controllare il nostro Arduino.

Per mettere in pratica quanto abbiamo imparato possiamo fare un gioco di prontezza di riflessi.

Come funziona il gioco:

è un gioco per due giocatori che hanno un pulsante e un LED rosso dalla propria parte. Al centro un LED blu. Quando si accende il LED blu il più veloce a premere il pulsante fa punto.

Per sapere chi ha fatto punto si guarda il LED rosso che si accende dal lato del vincitore.

Il gioco si ripete per 11 volte in modo che non possa finire in parità.

Questo programma potrebbe sembrare semplice ma presenta alcune difficoltà.

Il ragionamento logico potrebbe sembrare semplice:

- 1 - accendo il LED blu
 - 2 - leggo gli ingressi dei pulsanti
 - 3 - accendo il LED rosso dal lato del pulsante che si è chiuso per primo
 - 4 - aspetto un secondo
 - 5 - spengo il LED rosso
 - 6 - spengo il LED blu
 - 7 - aspetto un tempo
- e poi il programma ricomincia.

Non è così. Un programmatore è abituato a chiedersi sempre “**se**”.

Un esempio: vi state preparando per andare in spiaggia e come prima cosa mettete un telo da bagno e le infradito in borsa.

- ... e **se** poi mi viene sete? Aggiungete una bibita.
- ... e **se** poi mi viene fame? Aggiungete un pacco di biscotti.
- ... e **se** poi il sole mi infastidisce? Aggiungete un cappello.
- ... e **se** poi mi annoio? Aggiungete un libro.

PROGRAMMAZIONE - GIOCHIAMO :

È uguale per scrivere un gioco o per qualunque programma.

Nel nostro caso:

... e **se** un giocatore per fare il furbo preme in anticipo il suo pulsante?

... e **se** un giocatore non rilascia il pulsante dopo la giocata?

... e **se** un giocatore impara a memoria il tempo di attesa?

Provate ad immaginare quante domande di pongono i programmatore che scrivono il codice di una sonda spaziale che viaggia per tutto il sistema solare e che costa milioni di dollari.

Un micro controllore può prendere decisioni però non può trovare soluzioni se non sono già scritte nel suo codice, se tutto non è stato pensato e scritto nel codice una sonda spaziale si può perdere nel vuoto dell'universo.

Questo esempio serve per meglio comprendere il ragionamento logico che non si ferma sulla giusta serie di azioni che deve compiere il microcontrollore ma soprattutto cosa deve fare se accade una situazione imprevista come quella di un giocatore che prova a barare per vincere.

Ora che abbiamo compreso bene l'importanza del chiedersi sempre “**se**” nella programmazione e nella vita, analizziamo le domande che ci siamo posti.

Prima domanda:

... e se un giocatore per fare il furbo preme in anticipo il pulsante?

Si può fare che il punto sia del suo avversario.

... e se un giocatore non rilascia il pulsante dopo la giocata?

Non è un problema il punto sarà del suo avversario.

PROGRAMMAZIONE - GIOCHIAMO :

... e se un giocatore impara a memoria il tempo di attesa?

Si può fare che il tempo cambi in modo casuale.

Ponendoci domande ora abbiamo un altro ragionamento logico e il nostro gioco è migliore, oltretutto mi piacerebbe un gioco di luci che avvisa che il gioco sta per iniziare.

- 1 – faccio un gioco di luci per iniziare il gioco
- 3 – prendo un numero casuale per l'attesa
- 4 – aspetto il tempo casuale e controllo se un giocatore preme il suo pulsante
- 5 – se un pulsante passa a livello logico alto guadagna il punto l'avversario e prendo un altro numero. Se no:
- 6 – quando finisce il tempo di attesa accendo il LED blu
- 7 – controllo quale pulsante si chiude per primo e metto il punto al giocatore che ha premuto per primo.
- 8 – se la giocata è la numero 11 fermo il gioco e controllo chi ha vinto.

Sembra difficile? Realmente non lo è.

Avete tutte le conoscenze per scriverlo.

Questo programma sarà molto lungo. Non conviene scrivere un programma lungo tutto insieme. Come già scrisse un programma complesso è formato da programmi semplici.

È più comodo dividere il programma in pezzetti creando nuove funzioni. Nel linguaggio di Arduino, il C++, non abbiamo solo il “void setup()” e il “void loop()” ma possiamo creare nuove funzioni con il nome che vogliamo per scrivere dentro le parentesi graffe una parte del programma, per esempio: “void tempo()” che potrebbe essere la funzione che trova il numero casuale per l'attesa e che possiamo richiamare tutte le volte che ci serve.

Oltretutto dividere il programma in funzioni fa sì che se un giorno ci servisse una funzione che genera un numero casuale per un nuovo programma non la si debba riscrivere ma la si possa copiare e incollare da un altro programma.

PROGRAMMAZIONE - GIOCHIAMO :

Per fare questo gioco ci servono:

2 LED rossi

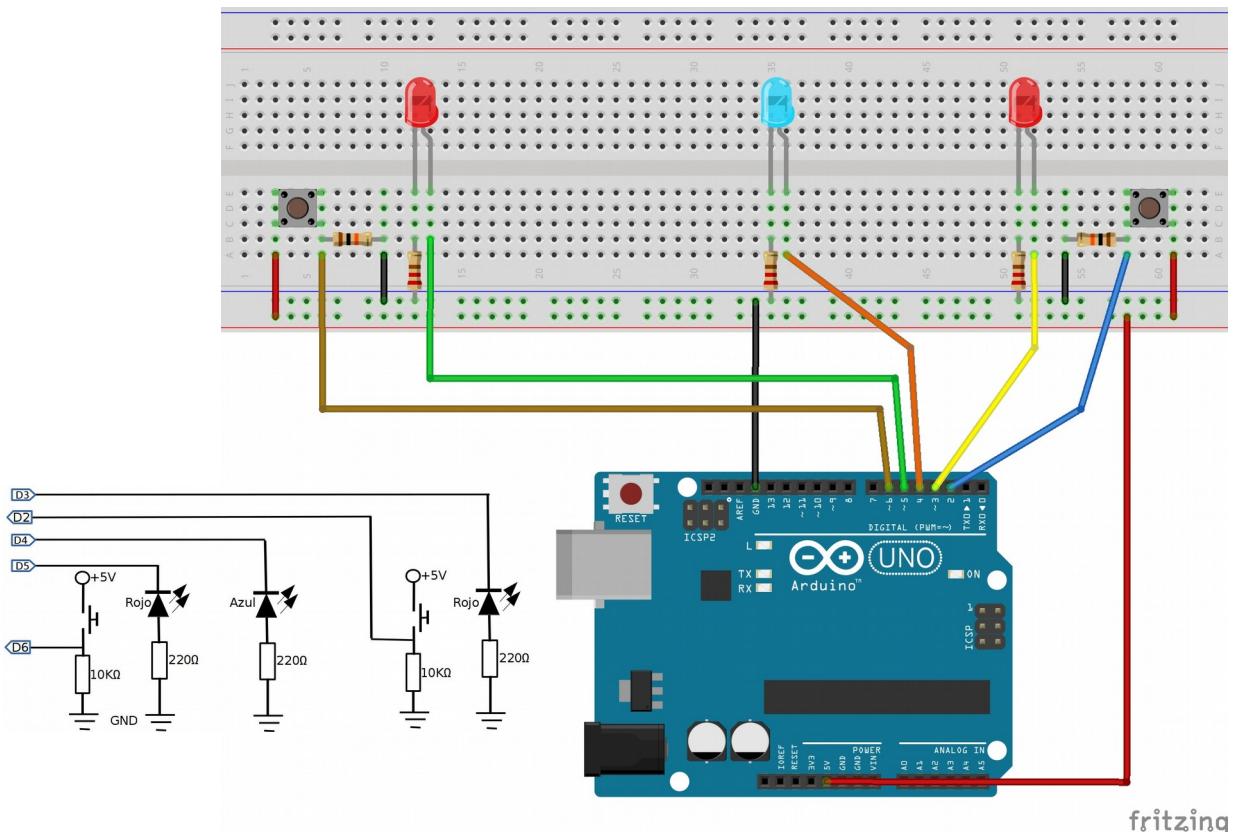
1 LED blu

3 resistenze da $220\ \Omega$ per i LED

2 pulsanti

2 resistenze da 10 K Ω (10,000 Ohm) per il pull down dei pulsanti.

Colleghiamo sulla basetta di prova come nello schema di cablaggio.



Ora possiamo scrivere il programma del nostro gioco.

Come prima cosa scriviamo le costanti per le connessioni e nel "setup()" impostiamo le uscite e gli ingressi.

PROGRAMMAZIONE - GIOCHIAMO :

```
1 #define pulsante_1 2
2 #define LED_1 3
3 #define LED_BLU 4
4 #define LED_2 5
5 #define pulsante_2 6

15 void setup() {
16   pinMode ( pulsante_1, INPUT);
17   pinMode ( LED_1, OUTPUT );
18   pinMode ( LED_BLU, OUTPUT );
19   pinMode ( LED_2, OUTPUT );
20   pinMode ( pulsante_2, INPUT );
21
22 }
23
```

Nel nome di una variabile non si possono usare accenti e spazi perché se mettiamo uno spazio per l'IDE sono due parole e una variabile non può avere due nomi. Però per noi è comodo avere dei nomi che rappresentino esattamente la variabile. Possiamo scrivere i nomi mettendo al posto dello spazio il trattino basso “under score” che si ottiene premendo sulla tastiera il maiuscolo e il meno. In questo modo l'IDE vede una sola parola e non va in errore.

Come si può notare se mettiamo nomi logici non serve quasi scrivere il commento.

Per creare una nuova funzione sotto la funzione “void loop()” dopo la parentesi graffa chiusa scriviamo:

```
void inizio_gioco(){
e premiamo l'"Enter" per mettere in automatico la parentesi graffa chiusa.
}
```

È importante abituarsi a chiudere subito la parentesi graffa perché se non chiudiamo una parentesi l'IDE non capisce quale gli manca e va in errore sull'ultima e poi diventiamo matti per trovare l'errore.

Abbiamo creato una funzione che si chiama “inizio_gioco”.

PROGRAMMAZIONE - GIOCHIAMO :

Quando nel “loop()” chiamiamo questa funzione il programma esegue le istruzioni contenute nella funzione.

In questo modo se dobbiamo fare molte volte un gruppo di istruzioni le scriviamo una sola volta.

Nella nuova funzione scriviamo il gioco di luci per iniziare il gioco che in questo caso dobbiamo eseguire una sola volta quando inizia il gioco quindi inseriamo una variabile di controllo che chiamiamo “inizio” e che impostiamo a zero.

Una variabile di controllo si usa per dare un limite a una parte del programma. Quando inizia il gioco la variabile è a zero, quando eseguiamo il gioco di luce la mettiamo a 1 in modo che non si possa ripetere fino al termine del gioco quando la rimerteremo a zero.

Sriviamo nel “loop()” un “if” come controllo di flusso in questo modo:

```
24 void loop(){
25   if (inizio==0){ // se è a zero è una nuova sfida
26     inizio_gioco(); // chiamo la funzione per il gioco di luci
27   }
28 }
29
```

così facendo se la variabile è a zero si può chiamare la funzione “inizio_gioco”, se la variabile è a 1 il programma passa oltre.

“inizio_gioco();” chiama la funzione, questo significa che Arduino ora esegue cosa è scritto nella funzione prima di continuare nel “loop()”.

Creiamo una variabile di tipo “byte” perché in questa variabile mettiamo solo uno zero o un uno e la chiamiamo “inizio”.

PROGRAMMAZIONE - GIOCHIAMO :

```

1 #define pulsante_1 2
2 #define LED_1 3
3 #define LED_BLU 4
4 #define LED_2 5
5 #define pulsante_2 6
6 byte inizio = 0; // quando inizia il gioco la variabile va a 1

```

Ora scriviamo tra le parentesi graffe della funzione “inizio_gioco” le istruzioni per fare il gioco di luce.

```

45 void inizio_gioco(){ // questa funzione fa il gioco di luci dell'inizio
46
47     inizio=1; // metto a 1 la variabile per non rifare il gioco di luci fino alla fine
48     for( int i=0; i<10; i++){ // ripeto 10 volte il gioco di luci
49         digitalWrite ( LED_1, HIGH );
50         delay (150);
51         digitalWrite ( LED_1, LOW );
52         digitalWrite ( LED_BLU, HIGH );
53         delay (150);
54         digitalWrite ( LED_BLU, LOW );
55         digitalWrite ( LED_2, HIGH );
56         delay (150);
57         digitalWrite ( LED_2, LOW );
58         digitalWrite ( LED_BLU, HIGH );
59         delay (150);
60         digitalWrite ( LED_BLU, LOW );
61     }
62     delay ( 1000 );
63     digitalWrite ( LED_1, HIGH );
64     digitalWrite ( LED_2, HIGH );
65     delay ( 3000 );
66     digitalWrite ( LED_1, LOW );
67     digitalWrite ( LED_2, LOW );
68
69 }
70

```

Carichiamo il programma in Arduino.

Il gioco di luce si esegue una volta sola perché la variabile di controllo “inizio” è a zero perché l'abbiamo messa a zero quando l'abbiamo creata e il controllo di flusso “if” che la vede uguale a zero chiama la funzione “inizio gioco”.

La prima linea della funzione mette a 1 la variabile e quando finisce di eseguirsi Arduino torna nel “loop()” e non può più eseguire la funzione.

Abbiamo scoperto come si creano nuove funzioni e come richiamarle.

PROGRAMMAZIONE - GIOCHIAMO :

Ora ci serve creare un tempo di attesa sempre diverso per il gioco in modo che i

giocatori non possano abituarsi.

Arduino non possiede fantasia però possiede un algoritmo che può generare numeri apparentemente casuali.

La funzione si chiama “**random** (minimo, massimo)” dove minimo è il numero più basso e massimo quello più alto che può generare.

valore = **random** (500, 5000);

In questo modo la funzione “ **random** ” mette nella variabile “valore” un numero compreso tra 500 e 5000 in modo che il gioco abbia un tempo di attesa da 500 a 5000 millisecondi prima di accendere il LED blu.

Tuttavia la sequenza di generazione dei numeri casuali è sempre la stessa. Per cambiare la successione dei numeri serve inserire un numero generatore che genererà una nuova sequenza.

Un buon metodo è quello di leggere un ingresso analogico collegato ad un pezzo di cavo elettrico. Il cavo elettrico si comporta come un'antenna ricevendo il rumore elettromagnetico dell'aria. Il valore della lettura del rumore lo usiamo come numero generatore.

La sintassi è:

randomSeed(numero); dove “numero” è il numero generatore.

nel nostro caso scriviamo :

randomSeed (analogRead(0));

il numero generatore sarà la lettura dell'ingresso analogico “0”.

Dato che il rumore elettromagnetico non è sempre uguale possiamo dire che i nostri numeri ora saranno sempre differenti.

Creiamo un'altra funzione sotto la funzione “inizio_gioco” nel programma.

PROGRAMMAZIONE - GIOCHIAMO :

```
71 void tempo(){ // questa funzione trova il tempo di attesa  
72  
73   randomSeed(analogRead(0)); // creo il numero generatore dei numeri casuali  
74   valore=random(500,5000); //memorizzo il numero casuale da 500 a 5000  
75 }  
76
```

Abbiamo inserito il numero casuale nella variabile “valore” ora dobbiamo creare la variabile sotto le altre nella parte alta dello sketch e creiamo una variabile di tipo “int” perché il numero sarà compreso tra 500 e 5000.

```
int valore = 0;
```

Nel “setup()” apriamo la comunicazione seriale come sappiamo:

```
Serial.print ( 9600 );
```

Ora nel “loop()” scriviamo sotto l'ultima linea l'istruzione per scrivere il numero generato sul monitor seriale.

Scriviamo:

```
tempo(); // chiamo la funzione tempo per generare un numero
```

poi:

```
Serial.println ( valore );
delay ( 500 );
```

Mettiamo un tempo di pausa di 500 millisecondi per avere il tempo di leggere i numeri generati.

Carichiamo il programma e apriamo il monitor seriale cliccando sopra la lente.
Al termine del gioco di luce potremo vedere i numeri che la funzione “random” sta generando.

Questi numeri saranno i tempi di attesa del gioco.

PROGRAMMAZIONE - GIOCHIAMO :

Ora che abbiamo visto come si generano i numeri casuali possiamo cancellare le linee con il “Serial.println” e con il “delay” che non ci servono più e andiamo oltre.

Creiamo una nuova funzione sotto l'ultima che abbiamo scritto che chiamiamo “aspetta_il_tempo(){“, in questa funzione andiamo a vedere se un giocatore nel tempo di attesa preme il pulsante.

Per generare l'attesa non possiamo usare la funzione “delay()” perché la funzione “delay()” blocca il microcontrollore e se è fermo non può fare nulla.

Esiste una funzione che si chiama “millis()” che legge un contatore che si trova dentro il microcontrollore che somma 1 tutti i millisecondi che passano da quando si alimenta Arduino. In questo modo possiamo leggere il valore del contatore, sommare il valore del tempo di attesa e aspettare che il valore del contatore si allinei con il valore della nostra somma. Quando sarà uguale o maggiore significa che è passato il tempo di attesa giusto.

Per esempio:

se prendiamo il numero del contatore che in quel momento è uguale a 15530 e noi abbiamo un tempo di attesa di 1500 millisecondi e sommiamo il tempo di attesa al valore del contatore otteniamo 17030. Ora dobbiamo continuare a leggere il valore del contatore che aumenta di uno ogni millisecondo e quando sarà uguale o maggiore della nostra somma vorrà dire che sono passati 1500 millisecondi, il tempo della nostra attesa.

Per contenere il numero del contatore dei millisecondi ci serve una variabile di tipo “unsigned long” perché è l'unica che possa memorizzare il contatore.

Per intenderci meglio:

```
attesa = millis(); // metto nella variabile attesa il numero del  
// contatore dei millisecondi
```

```
attesa = attesa + valore; //sommo il valore della variabile  
//“valore” alla variabile attesa.
```

PROGRAMMAZIONE - GIOCHIAMO :

Ora non ci resta che leggere per tutto il tempo dell'attesa i due pulsanti per vedere se un giocatore bara.

Per fare questo esiste un altro controllo di flusso che è simile al "if" che si chiama "**while**". Al contrario del "if" che esegue una sola volta le istruzioni nella sua parentesi, il "**while**" esegue sempre le istruzioni finché sarà vera la sua condizione.

In questo modo se mettiamo come condizione che il valore del contatore sia maggiore della nostra somma, il "**while**" per tutto il tempo che passa finché è vero controllerà i pulsanti.

Il "**while**" si usa molto per fermare un programma in attesa che una condizione si compia per esempio per aspettare che un motore termini la sua corsa o che una resistenza elettrica raggiunga una certa temperatura e molto altro.

```
77 void attesa_tempo() { // questa funzione controlla se un giocatore bara nell'attesa
78
79 attesa = millis(); // memorizzo il numero del contatore dei millisecondi
80 attesa = attesa + valore; // sommo al tempo letto del contatore il mio tempo di attesa
81
82 while ( attesa > millis() ){ // se la somma è minore del valore del contatore
83 if ( digitalRead (pulsante_1) == HIGH) { // guardo se il pulsante 1 è premuto
84 punti_2(); // se è premuto metto un punto al giocatore 2
85 rifare = 1; // porto a 1 la variabile di controllo per rifare la partita
86 }
87 if ( digitalRead (pulsante_2) == HIGH) { // guardo se il pulsante 2 è premuto
88 punti_1(); // se è premuto metto un punto al giocatore 1
89 rifare = 1; // porto a 1 la variabile di controllo per rifare la partita
90 }
91 }
92 }
```

PROGRAMMAZIONE - GIOCHIAMO :

- in linea 79 mettiamo nella variabile “attesa” il numero del contatore dei millisecondi.
- in linea 80 sommiamo alla variabile il valore della variabile “valore” che è il nostro tempo di attesa.
- in linea 82 il controllo di flusso “**while**” controlla i pulsanti valore del finché il tempo di attesa non diventa inferiore del pulsante 1 è contatore.
- in linea 83 con il controllo di flusso “**if**” controlliamo se il pulsante 1 è premuto. Se è premuto diamo un punto al giocatore 2.
- in linea 84 chiamo la funzione che più tardi andremo a scrivere e che chiameremo “punti_2” dove si memorizzano i punti fatti dal giocatore 2.
- in linea 85 metto un 1 nella variabile “rifare” che è una variabile di controllo. Se ha valore 1 il programma genera un numero per l’attesa come se fosse terminata la sfida in altro corso.
- la linea 87 fino alla 89 è identica alle precedenti solo che controlliamo il pulsante 2 che se è premuto diamo il punto al giocatore 1.

Ora andiamo in cima allo sketch e creiamo le variabili che abbiamo usato in questa funzione.

```
unsigned long attesa = 0; // memorizzo il tempo di attesa per la funzione "aspetta_il_tempo"
```

```
byte rifare = 0; // se "rifare" si porta a 1 è perché un giocatore ha barato.
```

Scriviamo la funzione che tiene il conto dei punti del giocatore 1 e del giocatore 2 che la funzione che abbiamo scritto chiama.

Il bello di scrivere in questo modo è che le funzioni possono chiamare altre funzioni. Se non scriviamo in questo modo ci troveremo costretti a scrivere molti controlli di flusso uno dentro l’altro: una follia.

PROGRAMMAZIONE - GIOCHIAMO :

La funzione la chiamiamo: “**void punti_1(){“**

```
94 void punti_1(){ // questa funzione aggiunge un punto al giocatore 1
95
96 punteggio_1 = punteggio_1 +1; // sommo 1 ai punti del giocatore 1
97
98 for ( int i=0; i<4; i++){ // faccio lampeggiare 4 volte il led del giocatore 1
99   digitalWrite ( LED_1, HIGH );
100   delay(200);
101   digitalWrite ( LED_1, LOW );
102   delay (200);
103 }
104 digitalWrite (LED_BLU, LOW); // spergo il led blu
105 }
106 }
```

- in linea 96 aggiungiamo un punto in più nella variabile “punteggio_1” che memorizza i punti del giocatore 1.
- in linea 98 scriviamo un controllo di flusso di tipo “for” per far lampeggiare il LED rosso del giocatore 1 che ha vinto un punto.
- in linea 104 spegnamo il LED blu nel caso fosse rimasto acceso.

Ora scriviamo la funzione per il punteggio del giocatore 2 che chiamiamo: “**void punti_2(){“**

```
107 void punti_2(){ // questa funzione aggiunge un punto al giocatore 2
108
109 punteggio_2 = punteggio_2 +1; // sommo 1 ai punti del giocatore 2
110
111 for ( int i=0; i<4; i++){ // faccio lampeggiare 4 volte il led del giocatore 2
112   digitalWrite ( LED_2, HIGH );
113   delay(200);
114   digitalWrite ( LED_2, LOW );
115   delay (200);
116 }
117 digitalWrite (LED_BLU, LOW); // spergo il led blu
118 }
119 }
```

PROGRAMMAZIONE - GIOCHIAMO :

Questa funzione è uguale all'altra cambia solo il nome della variabile e il LED quindi non sto a commentarla.

Andiamo a creare le variabili che abbiamo usato in queste due funzioni in testa al programma:

```
byte punteggio_1 = 0; // memorizzo i punti del giocatore 1  
byte punteggio_2 = 0; // memorizzo i punti del giocatore 2
```

nel “loop()” scriviamo la nuova parte dello sketch per fare un test delle funzioni.

```
24 void loop(){  
25   if (inizio==0){ // se è a zero è una nuova sfida  
26     inizio_gioco(); // chiamo la funzione per il gioco di luci  
27   }  
28   partite = partite +1; // sommo 1 per tenere il conto delle partite  
29   tempo(); // chiamo la funzione per il numero casuale dell'attesa  
30   attesa_tempo(); // chiamo la variabile che controlla che nessuno bari nell'attesa  
31 }  
32
```

- in linea 28 mettiamo un più 1 alla variabile “partite” che memorizza il numero delle giocate fatte che ci serve per finire il gioco alla numero 11.
- in linea 29 chiamiamo la funzione “tempo()” per generare il tempo di attesa.
- in linea 30 chiamiamo la funzione che controlla i pulsanti nel tempo di attesa per vedere se i giocatori barano.

Dobbiamo solo creare la variabile “partite” in testa allo sketch.

```
byte partite = 0; // in questa variabile memorizzo il numero delle partite giocate.
```

PROGRAMMAZIONE - GIOCHIAMO :

Possiamo caricare il programma nella memoria di Arduino.

Il “loop()” ora chiama la funzione per fare il gioco di luce dell'inizio, poi chiama la funzione per trovare un numero casuale che è il nostro tempo di attesa. Quando termina l'attesa ricomincia e dato che il gioco di luci lo ha già fatto, chiama nuovamente la funzione per trovare un altro numero casuale e spetta un'altra volta.

Se ora premiamo un pulsante fingendo di barare potremo vedere lampeggiare il LED dell'avversario che vuol dire che ha guadagnato un punto. Uguale se premiamo l'altro pulsante.

Scriviamo la funzione del gioco vero e proprio che chiamiamo:

“**void** competizione(){“

questa funzione deve solo controllare quale pulsante si chiude per primo, così potremo sapere qual'è il giocatore più veloce.

Anche in questo caso usiamo un controllo di flusso “while” che è specifico per aspettare un evento.

```
120 void gara(){ // questa funzione controlla chi preme per primo il pulsante
121
122     while ( gara_finita == 0 ){ // il "while" si ferma quando cambia di valore la variabile
123         if ( digitalRead ( pulsante_1 ) == HIGH ){ // se il giocatore 1 preme per primo
124             gara_finita = 1; // metto a 1 la variabile di controllo per fermare il "while"
125             punti_1(); // aggiungo un punto al giocatore 1
126         }
127
128         if (gara_finita == 0 ){ // se il giocatore 1 non ha premuto controllo il secondo
129
130             if ( digitalRead ( pulsante_2) == HIGH ){ // si el jugador 2 presiona por primero
131                 gara_finita = 1; // metto a 1 la variabile di controllo per fermare il "while"
132                 punti_2(); // aggiungo un punto al giocatore 2
133             }
134         }
135     }
136     gara_finita=0;
137 }
```

PROGRAMMAZIONE - GIOCHIAMO :

In questo caso nel controllo di flusso “while” mettiamo una variabile di controllo perché ci serve controllare due pulsanti e decidere cosa fare se si chiude uno o l’altro pulsante.

Creiamo una variabile di controllo che chiamiamo “competizione_finita” e che mettiamo a 1 quando uno dei due giocatori preme il pulsante.

- in linea 122 scriviamo il controllo di flusso “while” che esegue le istruzioni nelle sue parentesi graffe finché la variabile di controllo cambia di valore.
- in linea 123 scriviamo un controllo di flusso “if che controlla se il pulsante 1 è premuto.
- in linea 124 se il pulsante 1 è premuto portiamo a 1 la variabile di controllo in modo da fermare il controllo di flusso “while”.
- in linea 125 chiamiamo la funzione “punti_1()” per registrare il punteggio del giocatore 1.
- in linea 128 andiamo a controllare se la variabile di controllo è uguale a zero. Perché se il giocatore 1 ha premuto per primo non dobbiamo più leggere il pulsante 2 che sarà in ritardo rispetto al primo. Se la variabile di controllo è uguale a uno non consideriamo il pulsante 2.

Se il giocatore 1 non ha premuto:

- in linea 130 controllo se è premuto il pulsante 2.
- in linea 131 se è premuto metto la variabile di controllo a 1 per fermare il “while”.
- in linea 132 chiamo la funzione “punti_2” per aggiungere un punto al giocatore 2.
- in linea 136, fuori dalle parentesi graffe del controllo di flusso “while”, posso riportare a zero la variabile di controllo in modo che sia pronta per una nuova chiamata della funzione.

Creiamo la variabile di controllo in alto nel programma sotto le altre variabili.

```
byte competizione_terminata = 0; // va a uno quando un pulsante si preme nella competizione.
```

PROGRAMMAZIONE - GIOCHIAMO :

Dobbiamo solo scrivere una ultima parte del codice nel “loop()”.

```
24 void loop(){  
25     if (inizio==0){ // se è a zero è una nuova sfida  
26         inizio_gioco(); // chiamo la funzione per il gioco di luci  
27     }  
28     partite = partite +1; // sommo 1 per tenere il conto delle partite  
29     tempo(); // chiamo la funzione per il numero casuale dell'attesa  
30     attesa_tempo(); // chiamo la variabile che controlla che nessuno bari nell'attesa  
31     if (rifare==0){ // se "rifare" è = a 0 è perchè nessuno ha barato  
32         digitalWrite (LED_BLU,HIGH); // accendo il led blu per iniziare la sfida  
33         gara();  
34         if ( partite > 10){ // se il numero delle partite è 11 il gioco finisce  
35             fine(); // vado a vedere chi ha vinto  
36             punteggio_1 = 0; // metto a zero il punteggio del giocatore 1 per una nuova gara  
37             punteggio_2 = 0; // metto a zero il punteggio del giocatore 2 per una nuova gara  
38             inizio = 0; // metto a zero la variabile per iniziare un nuovo gioco  
39         }  
40     }  
41     rifare=0; //metto a zero la variabile per iniziare una nuova partita  
42 }  
43 }
```

Scriviamo un controllo di flusso di tipo “if” per vedere se il programma deve proseguire il gioco. Nella funzione “aspetta_il_tempo” avevamo messo una variabile di controllo che si chiamava “rifare” nel caso un giocatore avesse premuto il pulsante nel tempo di attesa, quindi se questa variabile è a zero possiamo proseguire e chiamare la funzione “competizione()”.

- in linea 31 se la variabile “rifare” è uguale a zero possiamo andare avanti con il gioco.
- in linea 32 accendiamo il LED blu.
- in linea 33 chiamiamo la funzione “competizione()”
- in linea 34 con un controllo di flusso “if” controlliamo se abbiamo già fatto 10 partite in questo caso
- in linea 35 chiamiamo la funzione “fine()” per vedere chi è il vincitore e per finire il gioco.

Dalla linea 36 alla 42 rimettiamo a zero tutte le variabili per una nuova competizione.

PROGRAMMAZIONE - GIOCHIAMO :

Scriviamo l'ultima funzione che si chiama “void fine() {}” e che termina il gioco.

```
138 void fine(){
139     if ( punteggio_1>punteggio_2){ // se il punteggio del giocatore 1 è maggiore
140         for(int i=0; i<4; i++){ // faccio lampeggiare il led blu e il led rosso
141             digitalWrite(LED_BLU,HIGH); // del giocatore 1 per 4 volte
142             digitalWrite(LED_1,HIGH);
143             delay(1000);
144             digitalWrite(LED_BLU,LOW);
145             digitalWrite(LED_1,LOW);
146             delay(500);
147         }
148     }
149
150     else { // se non è maggiore vince il giocatore 2
151         for(int i=0; i<4; i++){ // faccio lampeggiare il led blu e il led rosso
152             digitalWrite(LED_BLU,HIGH); // del giocatore 2 per 4 volte
153             digitalWrite(LED_2,HIGH);
154             delay(1000);
155             digitalWrite(LED_BLU,LOW);
156             digitalWrite(LED_2,LOW);
157             delay(500);
158         }
159     }
160     partite = 0; // metto a zero il numero delle partite giocate
161     delay ( 5000 ); // aspetto 5 secondi prima di ricominciare un nuovo gioco
162 }
163 }
```

- in linea 139 controlliamo se il punteggio 1 è maggiore del punteggio 2 se è maggiore vince il giocatore 1.
- in linea 140 con un controllo fi flusso “for” facciamo lampeggiare il LED blu e il LED rosso del giocatore 1.
- in linea 150 se non è maggiore è perché vince il giocatore 2.
- in linea 151 con un controllo di flusso “for” facciamo lampeggiare il LED blu e il LED rosso del giocatore 2.
- in linea 160 metto a zero la variabile che memorizza il numero delle giocate per una nuova sfida.

Prima di caricare lo sketch nella memoria di Arduino controlliamo se abbiamo creato tutte le variabili.

PROGRAMMAZIONE - GIOCHIAMO :

```
6 byte inizio = 0; // quando inizia il gioco la variabile va a 1
7 int valore=0; // in "valore" metto un numero casuale del tempo di attesa
8 unsigned long attesa = 0; // memorizzo il valore del contatore dei millisecondi per calcolare l'attesa
9 byte rifare = 0; // se un giocatore bara la variabile va a 1
10 byte punteggio_1 = 0; // memorizzo il punteggio del giocatore 1
11 byte punteggio_2 = 0; // memorizzo il punteggio del giocatore 2
12 byte gara_finita = 0; // va a 1 quando finisce una gara
13 byte partite = 0; // memorizzo il numero delle partite
14
```

Carichiamo il programma e giochiamo un po' per rilassarci.

Se osserviamo il programma scritto nel “loop()” possiamo vedere che è corto, e se leggiamo il commentario possiamo notare che è sufficiente un'occhiata per capire cosa fa il programma.

Scrivendo questo gioco non solo abbiamo imparato nuove istruzioni e nuovi controlli di flusso ma abbiamo fatto un passo avanti nel ragionamento logico.

Avevamo scritto un ragionamento logico una prima volta, dividendolo in parti più piccole ci siamo accorti che non avevamo pensato tutto. Questo perché dividere un problema è come osservarlo con la lente di ingrandimento.

ARDUINO - II DISPLAY LCD:

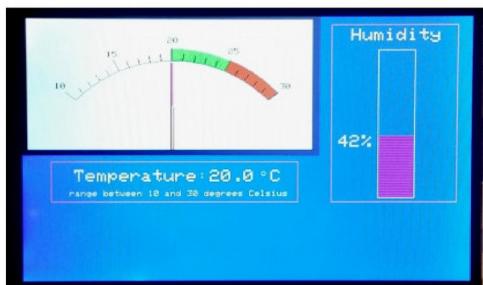
Esistono molti display che possiamo collegare a Arduino:



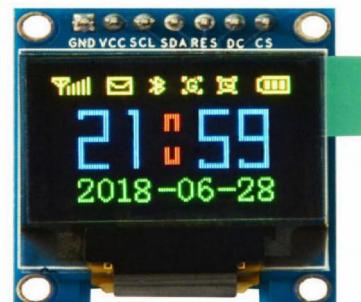
Display lcd 2 linee 16 caratteri



Display lcd 4 linee 20 caratteri



Display lcd grafico



Display oled grafico

Il primo display che si usa è il display lcd con due linee e 16 caratteri perché è semplice da usare ed è molto economico.

Questo tipo di display deve essere compatibile con Hitachi HD44780 che è il circuito integrato che gestisce il display e che è gestito dalla libreria.

Questo display ha 16 connessioni come si può vedere in figura.

ARDUINO - IL DISPLAY LCD:



- La connessione 1 (VSS) si collega a el GND di Arduino è il negativo del display.
- La connessione 2 (VDD) si collega al + 5 V di Arduino è il positivo del display.
- La connessione 3 (VE) è il contrasto e serve per regolare la visione del display
- La connessione 4 (RS) è il segnale di selezione del registro
- La connessione 5 (E) se positiva mette in modo lettura se negativa mette in modo scrittura il display
- Le connessioni da 6 a 13 sono il dato che può essere di 4 o 8 bit che il display scrive
- La connessione 14 è l'anodo + 5V del LED che illumina il display
- La connessione 16 è il catodo GND del LED che illumina il display

l'IDE ha una libreria per collegare il display che si chiama "LiquidCrystal" ma per collegare questo display ci servono 6 uscite digitali di Arduino che rubano spazio ai nostri progetti.

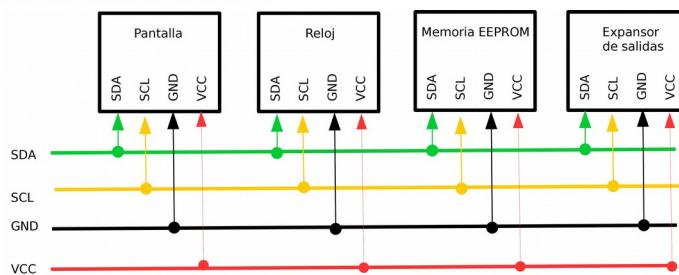
In vendita si trovano display che sotto il circuito hanno una scheda di conversione seriale-parallela che si collega al bus I2C.

ARDUINO - IL BUS I2C:



Arduino non ha solo la porta di comunicazione UART (RX-TX) che abbiamo usato per scrivere i dati sul monitor del computer ma ha anche una porta di comunicazione I2C.

Il bus I2C è una comunicazione seriale che può comunicare con molti device. Nel bus I2C tutti i device hanno un proprio indirizzo in questo modo con solo due fili e il GND possiamo comunicare con molti device.



Il bus di comunicazione I2C fu inventato da Philips nel 1982 ed è uno standard di comunicazione per circuiti integrati che funziona con un Master e molti Slave. Nel nostro caso Arduino è il Master e tutte le schede che colleghiamo sono Slave. Tutti gli Slave hanno un proprio indirizzo, non si possono collegare Slave con lo stesso indirizzo. L'indirizzo si identifica con un numero a due cifre di tipo esadecimale.

Tutti gli Slave si collegano in parallelo generando un bus, il SDA e il SCL sono i due segnali che trasmettono e ricevono i dati. Il Master (Arduino) invia sul bus un bit di START e poi l'indirizzo dello Slave col quale chiede di comunicare, il Master decide se leggere o scrivere dati e lo Slave legge o scrive i dati come richiesto dal Master.

Abbiamo già visto la numerazione decimale e binaria, la prima ha 10 numeri per cifra mentre la seconda solo due.

La numerazione esadecimale ha 16 numeri per cifra.

Ma come si possono scrivere 16 numeri per cifra se i numeri sono 10?

ARDUINO - IL BUS I2C:

La numerazione esadecimale mette dopo il 9 le lettere da A maiuscola a F maiuscola.

| Esadecimale | Decimale | Binario |
|-------------|----------|---------|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

In questo modo una sola cifra può rappresentare un Byte di 4 bit.

Per conoscere che numero decimale rappresenta il numero esadecimale FF moltiplichiamo per 16 la prima cifra che è F e ha un valore di 16.

$$15*16=240$$

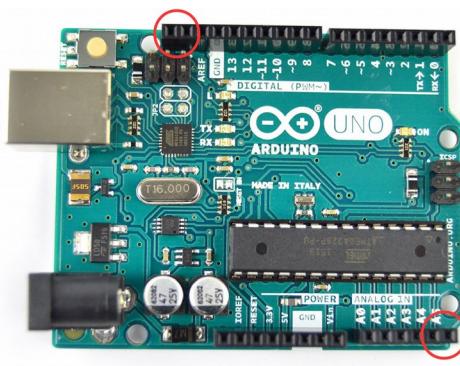
Poi sommiamo il valore della seconda cifra che è F e ha un valore di 16.

$$240+16=256$$

e otteniamo 256 che è un Byte di 8 bit.

$$\text{FF} = 255 = 11111111$$

Con solo due numeri possiamo scrivere 256 indirizzi ma il protocollo della comunicazione I2C usa solo 7 bit per gli indirizzi quindi il numero massimo di device collegabili è 128.



Come avviene per la comunicazione UART che trasforma le connessioni 0 e 1 in RX e TX, lo stesso avviene con la comunicazione I2C che trasforma in SCL e SDA l'ingresso analogico A4 e A5.

Con Arduino Uno possiamo scegliere dove collegare il bus se in A4 e A5 o su due connessioni dedicate. In altri modelli di Arduino come l'Arduino Nano uguale ad Arduino Uno solo più piccolo non troviamo connessioni dedicate e quindi siamo obbligati a collegarci agli ingressi analogici.

PROGRAMMAZIONE - LE LIBRERIE:

Per usare il bus di comunicazione I2C ci serve la libreria "Wire".

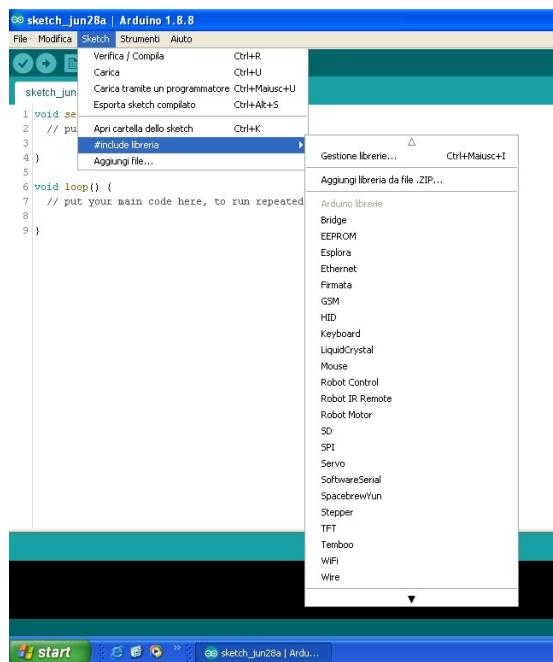
Per spiegare cosa sono le librerie vi faccio un esempio:

Vi ricordate il film MATRIX ? In questo film Neo quando aveva la necessità di apprendere nuove cose si inseriva un connettore nel cervello collegato ad un computer e in pochi secondi imparava il Kung-fu o a pilotare un elicottero.

Una libreria funziona in modo simile. Dota di abilità extra il nostro programma.

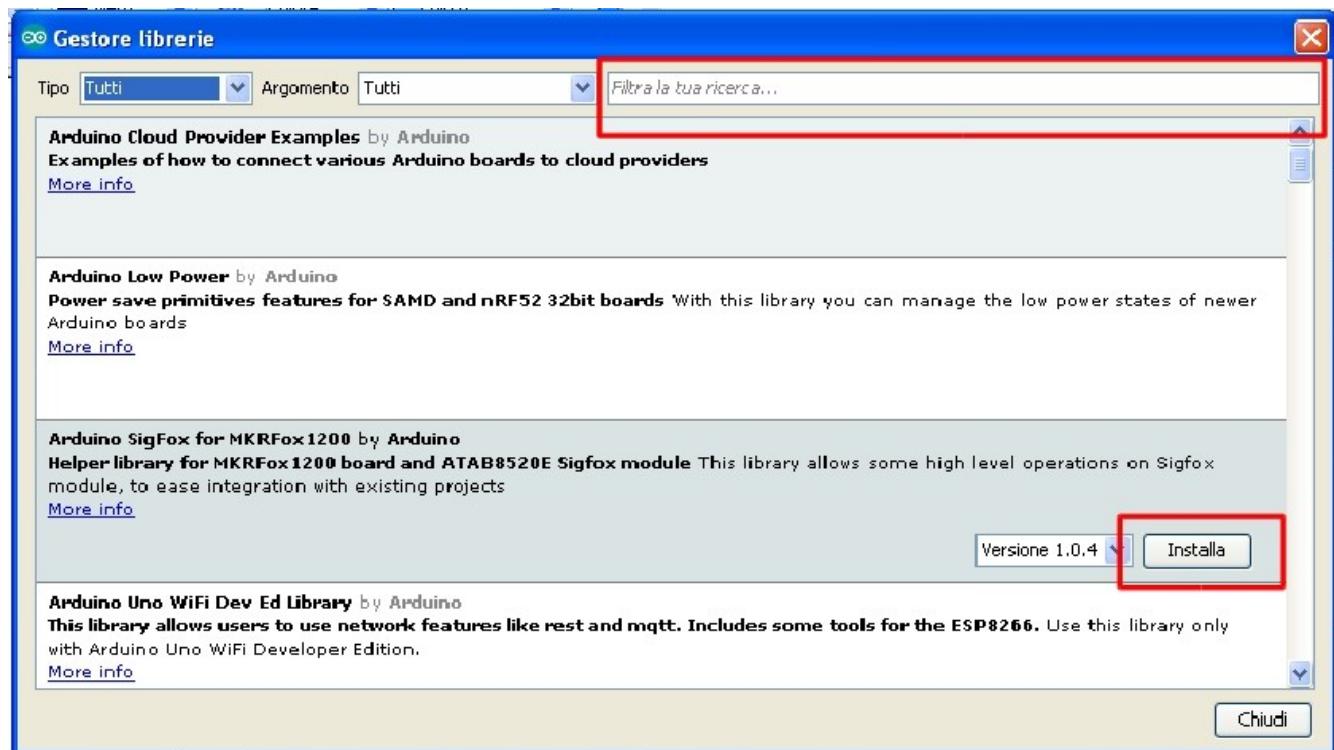
Per usare componenti elettronici esterni come display, sensori o servomotori dobbiamo studiare molto bene i componenti per sapere come gestirli e non è semplice farlo soprattutto per chi è all'inizio e sta imparando. Esistono programmi che sono stati scritti da altri sviluppatori che studiarono bene i componenti e che misero a disposizione di tutti i loro programmi. Questi programmi si chiamano librerie e sono scritte con una struttura specifica in modo che l'IDE possa inserirli nei nostri Sketch.

Per vedere le librerie che sono inserite nel IDE dalla barra dei menù clicchiamo su "Sketch" e poi su "#include libreria".



PROGRAMMAZIONE - LE LIBRERIE:

Nella finestra si possono vedere tutte le librerie che possiede l'IDE. Se non vediamo la libreria che stiamo cercando nella prima riga della finestra possiamo vedere "Gestione librerie" e se ci clicchiamo sopra si apre una nuova finestra dove possiamo cercare la libreria del componente che ci serve se esiste.



Nella finestra di ricerca scriviamo il nome del componente elettronico che stiamo cercando o il nome della libreria. Quando lo troviamo clicchiamo su "Install", ora la libreria si può vedere nella lista delle librerie installate nell'IDE.

PROGRAMMAZIONE - LE LIBRERIE:

Per usare il bus I2C dobbiamo caricare la libreria che si trova nell'elenco delle librerie dell'IDE che si chiama "Wire".

Nella barra del menù clicchiamo su "Sketch" e poi, nel menù che si apre, clicchiamo su "#include libreria" e nel nuovo menù clicchiamo sopra a "Wire".

Ora possiamo vedere che nella prima linea dello Sketch si trova la libreria.

```
sketch_jun28a §
1 #include <Wire.h>
2
3 void setup() {
4     // put your setup code here, to run once:
5
6 }
7
8 void loop() {
9     // put your main code here, to run repeatedly:
10}
11}
```

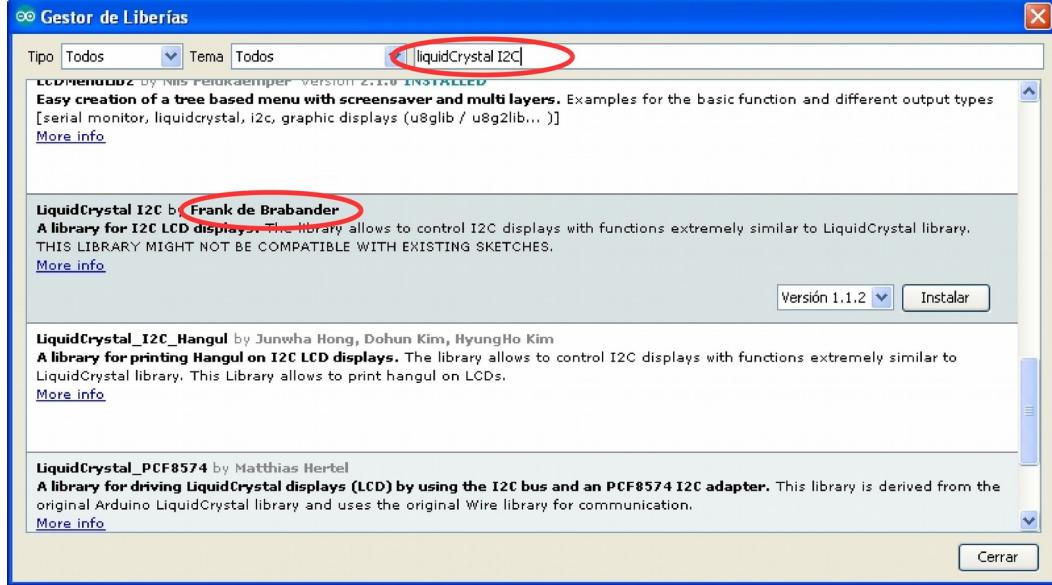
"#include" comunica all'IDE di caricare il codice della libreria nel nostro Sketch. Con il simbolo cancelletto "#" comunichiamo al pre processatore del IDE come già avevamo fatto con l'istruzione "#define" per dire al pre processatore di trovare e sostituire le costanti che avevamo usato per le connessioni, con "#include" diciamo di includere un programma esterno nel nostro prima di convertire lo Sketch in linguaggio macchina.

Ora ci serve la libreria per il display I2C che non l'abbiamo in lista.

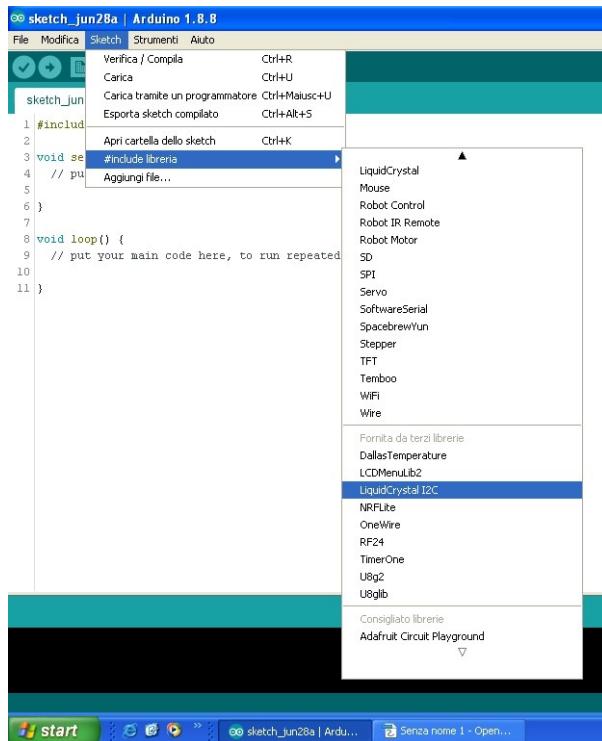
Nella finestra dalla quale abbiamo caricato la libreria "Wire" la prima voce in alto è "Gestione librerie", clicchiamola e si apre la finestra.

In alto nel campo di ricerca scriviamo "liquidCrystal I2C", attendiamo qualche secondo e vedremo apparire le librerie che corrispondono alla chiave di ricerca. Scegliamo quella scritta da Frank de Bradander.

PROGRAMMAZIONE - LE LIBRERIE:



installiamo la libreria e chiudiamo la finestra. Adesso la nuova libreria è pronta e la possiamo vedere tra le librerie del IDE.



PROGRAMMAZIONE - LE LIBRERIE:

Carichiamo la libreria e ora nel IDE abbiamo due librerie.

```
1 #include <Wire.h>
2
3 #include <LiquidCrystal_I2C.h>
4
5 void setup() {
6     // put your setup code here, to run once:
7
8 }
9
10 void loop() {
11     // put your main code here, to run repeatedly:
12
13 }
```

Per conoscere l'indirizzo di un device I2C ci serve uno scanner I2C.

Lo troviamo in internet nel sito di Github.

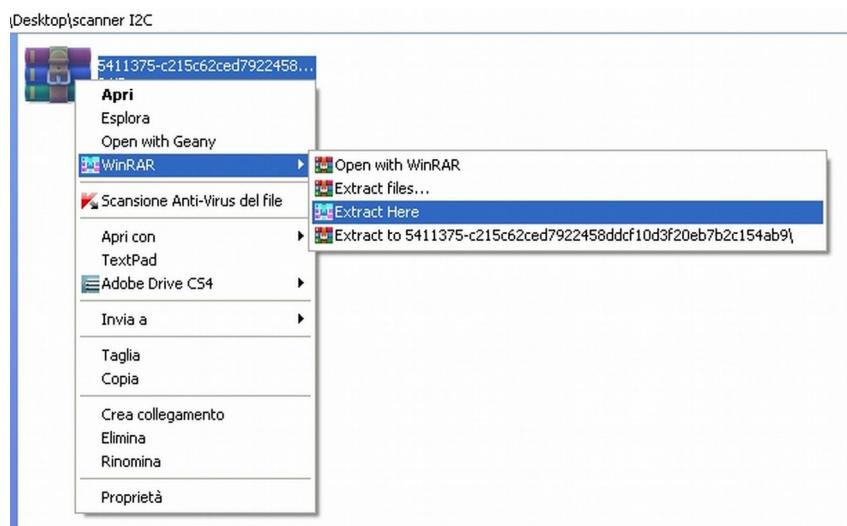
<https://gist.github.com/tfeldmann/5411375>

GitHub è un servizio pubblico di gestione del codice sorgente basato sul sistema Git, inventato da Linus Torvalds anima e per anni programmatore di Linux.

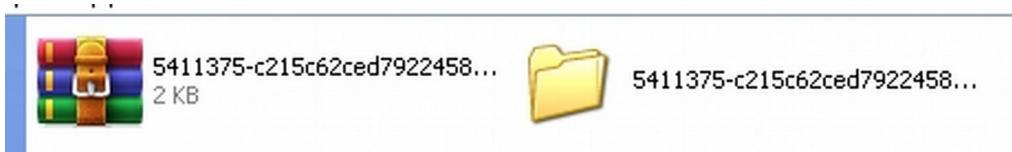
PROGRAMMAZIONE - LE LIBRERIE:

Per scaricare il programma clicchiamo sul bottone “Download ZIP”.

Ora cerchiamo la cartella compressa nella cartella dove l'ha salvata il computer e la copiamo e incolliamo in una nuova cartella nel desktop. Per decomprimere il file ci clicchiamo sopra col tasto destro del mouse come si vede in figura.

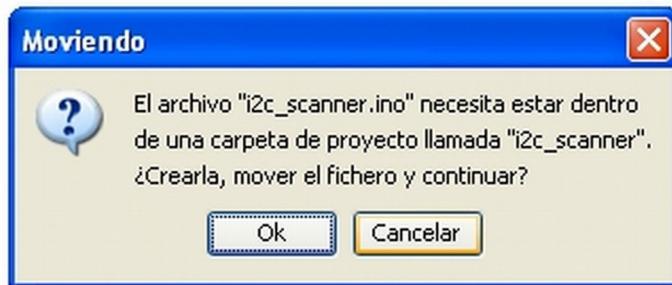


Ora che abbiamo decompresso il file e vediamo la cartella passiamo a aprire il programma dello scanner con l'IDE.



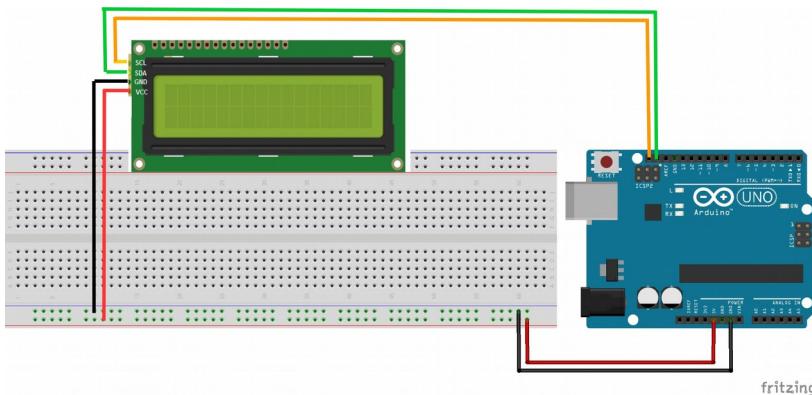
PROGRAMMAZIONE - LE LIBRERIE:

Quando chiediamo di aprire lo Sketch che è contenuto nella cartella l'IDE ci segnala un errore perché per l'IDE lo sketch e la cartella devono avere lo stesso nome.



Confermiamo con “OK”. L'IDE ora crea una cartella con il nome corretto e salva lo Sketch. Adesso possiamo copiare la cartella con il nome corretto e conservarla per future applicazioni.

Ora che abbiamo tutti gli strumenti necessari possiamo collegare il display a Arduino per cercare il suo indirizzo.



Carichiamo il programma dello scanner I2C in Arduino e apriamo il monitor seriale.

Il programma dello scanner ci comunica dove ha trovato un device che ha risposto, nel mio caso ha risposto all'indirizzo 3F. Voi potreste avere un altro indirizzo perché i display non hanno tutti lo stesso indirizzo.

PROGRAMMAZIONE - LE LIBRERIE:

Nel nostro programma dobbiamo scrivere sotto la linea che include la libreria del display I2C che tipo di display stiamo collegando e il suo indirizzo.

La sintassi è:

```
LiquidCrystal_I2C lcd( 0x"indirizzo", "numero caratteri per linea", "numero di linee" )
```

scriviamo le caratteristiche del display.

```
LiquidCrystal_I2C lcd(0x3F, 16, 2);
```

(Se avete un altro indirizzo scrivete il vostro indirizzo e nel caso usiate un altro display i dati del vostro display.)

Nel “Setup()” dobbiamo inizializzare il display con questa sintassi:

```
lcd.init();
```

e accendere il LED che illumina il display:

```
lcd.backlight();
```

Ora il display è pronto.

```
sketch_jun28a §
1 #include <Wire.h>
2
3 #include <LiquidCrystal_I2C.h>
4
5 LiquidCrystal_I2C lcd( 0x3F, 16, 2 ); // inserisco i dati del display
6
7 void setup() {
8
9 lcd.init(); // inizializzo il display
10 lcd.backlight(); // accendo il LED del display
11 }
12
13 void loop() {
14 // put your main code here, to run repeatedly:
15
16 }
```

PROGRAMMAZIONE - LE LIBRERIE:

Per scrivere sul display come prima cosa dobbiamo decidere dove posizionare il cursore.

```
lcd.setCursor(posizione, linea);
```

Dove la posizione è il punto dove si pone il cursore e si inizia a scrivere il primo carattere, la linea è la linea nella quale andiamo a scrivere. Il punto in alto a sinistra è il punto 0,0 perché i numeri iniziano da zero.

Per scrivere sul display si usa l'istruzione:

```
lcd.print("Hello, World");
```

se vogliamo scrivere una stringa alfanumerica ovvero una parola o una frase,

```
lcd.print(variabile);
```

se vogliamo scrivere il valore di una variabile.

È come usare il `Serial.print` solo che per il display scriviamo `lcd` al posto di `Serial`.

Per fare una prova scriviamo nel “loop()”:

```
lcd.setCursor(1, 0);
lcd.print("Hello, World");
lcd.setCursor(0, 1);
lcd.print("16x2 LCD Screen");
```

in questo modo in posizione 2 (perché inizia da zero) scriviamo “ `Hello, World` ” e sotto in posizione 1 scriviamo “ `16x2 LCD Screen` ”



PROGRAMMAZIONE - LE LIBRERIE:

```
1 #include <Wire.h>
2
3 #include <LiquidCrystal_I2C.h>
4
5 LiquidCrystal_I2C lcd( 0x3F, 16, 2 ); // inserisco i dati del display
6
7 void setup() {
8
9 lcd.init(); // inizializzo il display
10 lcd.backlight(); // accendo il LED del display
11 }
12
13 void loop() {
14 lcd.setCursor ( 1, 0 );
15 lcd.print ( "Hello, World" );
16 lcd.setCursor ( 0, 1 );
17 lcd.print ( "16x2 LCD Screen" );
18 }
19 }
```

Con un display è molto comodo scrivere programmi che informano sullo stato di un sensore o di un punto macchina e si possono creare dei menù.

Per realizzare una applicazione interessante con il display possiamo creare un ciclo computer in grado di comunicare la velocità in Km/h, il tempo del viaggio, la temperatura esterna e regolare la luminosità del display in modo che se pedaliamo la sera non ci abbagli.

Come sempre scriviamo il programma dividendolo in parti.

PROGRAMMAZIONE – CICLO-COMPUTER:

Quando scriviamo un programma che fa uso del display come prima cosa è logico pensare a come fare la grafica.

Nel “loop()” scriviamo la parte grafica per il display, nella prima linea possiamo scrivere la velocità e la temperatura e nella seconda linea il tempo di viaggio.



Kmh 00 t 00.00
h 00:00:00

Apriamo un nuovo Sketch nel IDE e salviamo come “ciclo computer”.

In alto prima del “setup()” scriviamo le chiamate delle librerie e le variabili.

```
1
2 #include <Wire.h>
3 #include <LiquidCrystal_I2C.h>
4
5 LiquidCrystal_I2C lcd(0x3F,16,2); // metto i dati del display
6
7 byte velocita=0; // memorizzo la velocità in kmh
8 float gradi=0; // memorizzo la lettura della temperatura
9 byte ore=0; // memorizzo le ore di viaggio
10 byte minuti=0; // memorizzo i minuti di viaggio
11 byte secondi =0; // memorizzo i secondi di viaggio
12
13 void setup()
14 {
15   lcd.init(); // inizializzo il display
16   lcd.backlight(); // accendo il led del display
17
18 }
```

Le variabili contengono i dati che scriveremo sul display per la velocità, la temperatura e il tempo di percorrenza. Per la temperatura usiamo una variabile di tipo “float” in modo che sia più precisa.

PROGRAMMAZIONE – CICLO-COMPUTER:

Nel “loop()” scriviamo il codice che gestisce il display.

```
21 void loop()
22 {
23     lcd.setCursor(0, 0);          // metto il cursore in alto a sinistra
24     lcd.print("Kmh ");           // scrivo Kmh più uno spazio
25
26     if (velocita<10) lcd.print("0"); // se la velocità è minore di 10 scrivo uno 0
27     lcd.print(velocita);          // scrivo il valore della velocità
28     lcd.setCursor(9,0);          // metto il cursore in posizione 10 nella prima linea
29     lcd.print("t ");             // scrivo "t" più uno spazio
30     if (gradi<10) lcd.print("0"); // se gradi è minore di 10 scrivo uno 0
31     lcd.print(gradi);           // scrivo i gradi
32     lcd.setCursor(3,1);          // metto il cursore in posizione 3 basso
33     lcd.print("h ");             // scrivo "h" più uno spazio
34     if (ore<10) lcd.print("0"); // se il valore delle ore è minore di 10 scrivo uno 0
35     lcd.print(ore);              // scrivo il valore delle ore
36     lcd.print(":");             // scrivo ":"
37     if (minuti<10) lcd.print("0"); // se il valore dei minuti è minore di 10 scrivo uno 0
38     lcd.print(minuti);           // scrivo il valore dei minuti
39     lcd.print(":");             //scrivo ":" 
40     if (secondi<10) lcd.print("0"); // se il valore dei secondi è minore di 10 scrivo uno 0
41     lcd.print(secondi);          // scrivo il valore dei secondi
42 }
```

Come si può vedere non è sempre necessario scrivere “lcd.setCursor”.

Lo scriviamo solo se dobbiamo mettere il cursore in una posizione precisa per scrivere perché se dobbiamo scrivere di seguito non serve dichiararlo perché il cursore si sposta in avanti.

Un'altra cosa interessante è come ho scritto il controllo di flusso “if” che non ha le parentesi graffe. Se il controllo di flusso deve eseguire solo un'istruzione si può scriverla in sequenza terminandola sempre col punto e virgola. In questo modo è più veloce scrivere.

Ho messo il controllo di flusso “if” per scrivere uno zero se le variabili sono minori di 10 perché se scriviamo una variabile inferiore a 10 il display scrive solo 5 ma noi vogliamo vedere il tempo come in un orologio e quindi ci serve che scriva 05 per vedere il tempo rappresentato così 01:05:09 e non così 1:5:9 che è brutto.

PROGRAMMAZIONE – CICLO-COMPUTER:

Ci serve conoscere:

- il tempo di percorrenza
- la temperatura
- la velocità
- regolare la luminosità del display

Scriviamo la funzione per calcolare il tempo di percorrenza.

Per fare questo usiamo la funzione "millis()" che conosciamo.

Mettiamo il valore di "millis()" già diviso per 1000 in una variabile di tipo "unsigned long" che chiamiamo "tempo" in modo d'avere il tempo in secondi. Tuttavia a noi serve conoscere le ore, i minuti e i secondi.

Per ottenere le ore dividiamo il valore della variabile "tempo" per 3600 perché un'ora è formata da 3600 secondi.

Se il risultato è minore di 1 vuol dire che non ho un numero di secondi sufficiente per fare un'ora e sul display scriverò "00".

Faccio un esempio:

tempo è uguale a 3000 secondi.

$$3000 \text{ secondi} / 3600 = 0.833333$$

però per memorizzare questo numero mi serve una variabile di tipo "float" che può memorizzare numeri con la virgola che chiamo "tmp" come il computer chiama una variabile temporale perché questa variabile la userò solo per fare questo calcolo.

$$\text{Tmp} = \text{tempo} / 3600;$$

Se ora memorizzo la variabile "tmp" nella variabile "ore" che è una variabile di tipo "int" si memorizzerà solo lo zero perché la variabile "int" non memorizza i numeri dopo la virgola.

Se il valore del tempo è uguale a 12015 secondi:

$$12,000 / 3,600 = 3.3375$$

Quando memorizzerò questo numero nella variabile "ore" si memorizzerà il valore 3.

Come abbiamo fatto per calcolare i numeri binari usando i pesi dei numeri facciamo la stessa cosa per calcolare il tempo di viaggio.

PROGRAMMAZIONE – CICLO-COMPUTER:

Se ora metto nella variabile “tmp” il valore della variabile tempo meno la variabile “ore” moltiplicata per 3600 ottengo il numero dei secondi che compongono i minuti.

`tmp = tiempo - (horas * 3600);`

La variabile “ore” contiene il numero 3

$$3 * 3,600 = 10,800$$

Quando sottraggo questo valore alla variabile “tempo”:

$$12,015 - 10,800 = 1215 \text{ La variabile “tempo” ha questo valore.}$$

Nella variabile “tmp” metto il valore della variabile “tempo” divisa per 60 perché per fare un minuto servono 60 secondi.

`tmp= tempo / 60;`

Se divido il valore della variabile tempo per 60 ottengo il numero dei minuti.

$$1,215 / 60 = 20,25$$

Metto il valore nella variabile “minuti” che è di tipo “int” e non memorizza i decimali.

`Minuti = tmp;`

La variabile “minuti” ora ha un valore di 20

`secondi = tempo - (minuti * 60);`

$$20 * 60 = 1,200$$

tempo ha un valore di 1,215

$$1,215 - 1,200 = 15 \text{ che sono i secondi.}$$

PROGRAMMAZIONE – CICLO-COMPUTER:

Scriviamo il programma.

Creiamo le due variabili nella parte alta dello Sketch:

```
float tmp = 0; // è una variabile temporale per il calcolo delle ore, minuti, secondi  
long tempo = 0; // memorizza in secondi il tempo di viaggio
```

Possiamo scrivere la funzione che chiamiamo “tempo_di_viaggio”.

```
48 void tempo_di_viaggio(){      // calcolo il tempo di viaggio  
49   tempo = millis() / 1000; // memorizzo il tempo in secondi  
50   tmp = tempo / 3600; // divido i secondi per 3,600 per trovare le ore  
51   ore = tmp; // memorizzo la parte intera della divisione  
52   tempo = tempo - (ore * 3600); // moltiplico la parte intera per 3,600 e la sottraggo al totale dei secondi per trovare i minuti  
53   tmp = tempo / 60; // divido i secondi restanti per 60 per calcolare i minuti  
54   minuti = tmp; // memorizzo la parte intera del calcolo dei minuti  
55   secondi = tempo - (minuti * 60); // moltiplico i minuti per 60 e la sottraggo dal totale per trovare i secondi  
56  
57 }  
58
```

Nel “loop()” sotto l'ultima linea di codice scriviamo la chiamata della funzione:

```
tempo_di_viaggio();
```

Carichiamo il programma in Arduino e possiamo vedere il tempo di viaggio che avanza sul display.

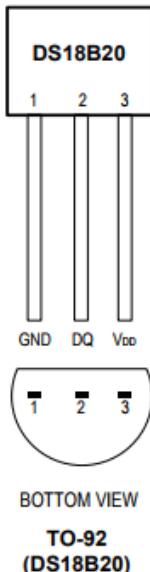
Ora ci serve misurare la temperatura. Per farlo ci serve un sensore di temperatura.

Il sensore più semplice e preciso è il Dallas DS18B20.

ARDUINO – SENSORE DI TEMPERATURA:

Per misurare la temperatura ci serve un sensore di temperatura.

Il DS18B20 può misurare temperature comprese tra -55°C y 125°C.



Nella figura si può vedere il sensore nella versione TO-92. In elettronica la forma dei componenti è identificata da un codice. In questo caso si chiama TO-92, questo perché un componente potrebbe avere forme differenti.

Il DS18B20 può essere alimentato da 3 a 5 Volt.

Il terminale 1 è la massa (GND).

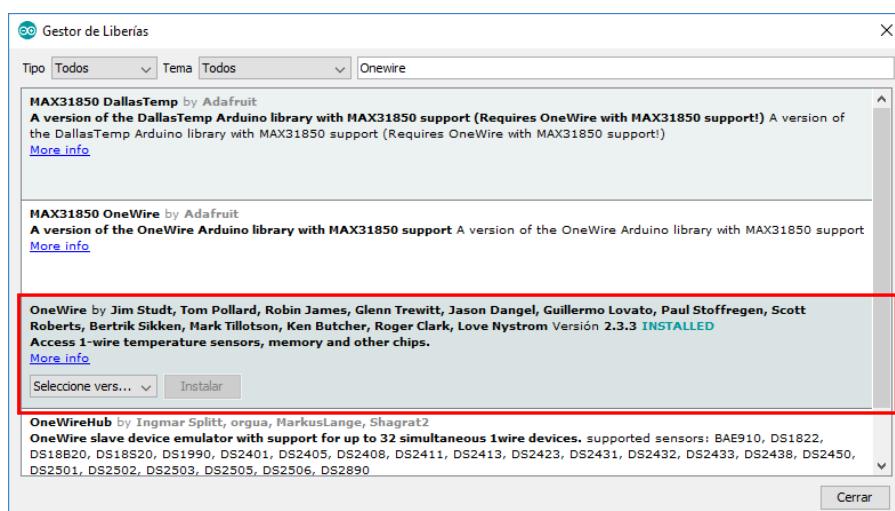
Il terminale 3 è il positivo i 5 V di Arduino.

Il terminale 2 è il DQ la porta di comunicazione.

Il DS18B20 è un sensore digitale che comunica con il protocollo "OneWire" che vuol dire "un filo".

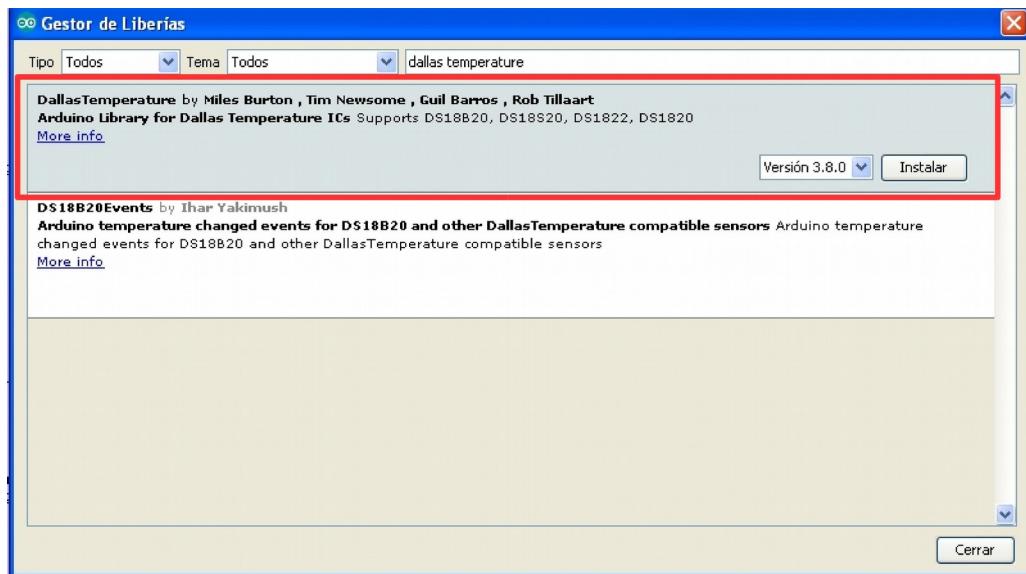
Per 'parlare' col sensore ci serve la libreria "OneWire" che si trova nel gestore delle librerie.

Andiamo a caricare la libreria che ci serve.



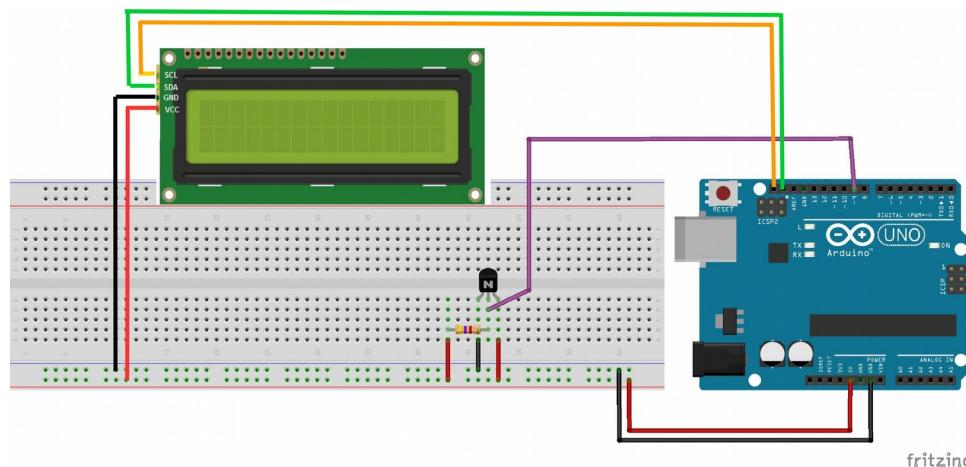
ARDUINO – SENSORE DI TEMPERATURA:

Dobbiamo scaricare anche la libreria che gestisce il Dallas DS18B20 che la troviamo sempre nel gestore delle librerie mettendo come chiave di ricerca “dallas temperature”.

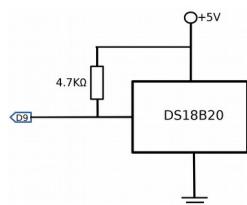


Scegliamo quella di Miles Burton.

Collegiamo DS18B20 come in figura.



ARDUINO – SENSORE DI TEMPERATURA:



La resistenza di pull-up è da 4.7 KΩ (4700 Ohm - giallo, viola e rosso se è con 4 anelli oppure giallo, viola nero e marrone se ha 5 anelli) e serve come pull-up per la trasmissione digitale dei dati.

Ora possiamo inserire la libreria “DallasTemperature” dal menù “Sketch” - “#include libreria” che si metterà in alto nel programma e dopo includiamo la libreria “OneWire” che si metterà sopra.

```
1 #include <OneWire.h>
2 #include <DallasTemperature.h>
3 #define DQ 9 // è la connessione del DQ del DS18B20
4 OneWire oneWire(DQ); // comunica a la libreria "oneWire" la connessione del DS18B20
5 DallasTemperature sensorDS18B20(&oneWire); // comunica a la libreria "DallasTemperature" la connessione del DS18B20
6
7
8 #include <Wire.h>
9 #include <LiquidCrystal_I2C.h>
```

Sotto alle nuove librerie scriviamo nella linea 3 la definizione di QT (QT è il nome del terminale centrale del DS18B20) che abbiamo collegato all'ingresso 2 di Arduino.

La linea 4 comunica alla libreria dove trovare il sensore.

La linea 5 comunica all'altra libreria dove si trova il sensore.

Scriviamo la funzione per leggere la temperatura che chiamiamo:
void temperatura (){

```
66 void temperatura(){
67   sensorDS18B20.requestTemperatures(); // voy a solicitar la lectura de la temperadura a la librera
68   grados = sensorDS18B20.getTempCByIndex(0); // pongo en "grados" la lectura del dado de la librera
69 }
70 }
```

ARDUINO – SENSORE DI TEMPERATURA:

Come si può vedere tutto il lavoro viene svolto dalle librerie.

In linea 67 chiediamo la lettura della temperatura.

In linea 68 memorizziamo il dato letto nella variabile “gradi”.

Nell'ultima linea del “loop()” scriviamo la chiama della funzione che abbiamo appena scritto.

```
temperatura();
```

```
42 if (ore<10) lcd.print("0");// se il valore delle ore è minore di 10 scrivo uno 0
43 lcd.print(ore);           // scrivo il valore delle ore
44 lcd.print(":");         // scrivo ":" 
45 if (minuti<10) lcd.print("0");// se il valore dei minuti è minore di 10 scrivo uno 0
46 lcd.print(minuti);      // scrivo il valore dei minuti
47 lcd.print(":");         //scrivo ":" 
48 if (secondi<10) lcd.print("0");// se il valore dei secondi è minore di 10 scrivo uno 0
49 lcd.print(secondi);     // scrivo il valore dei secondi
50 tempo_di_viaggio();    // chiamo la funzione "tempo di viaggio"
51 temperatura();          // chiamo la funzione per leggere la temperatura
52
53 }
54 }
```

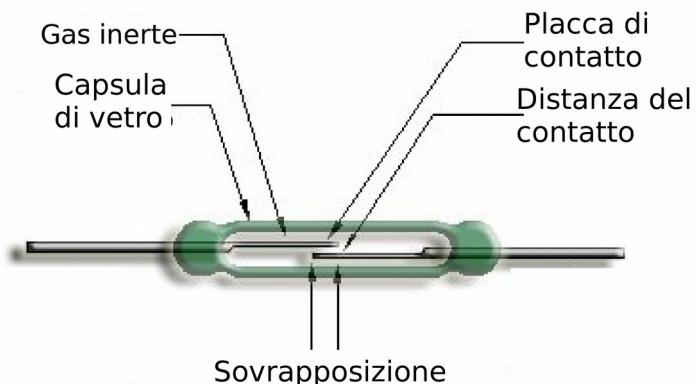
Possiamo caricare il programma in arduino e vedere la temperatura ambiente sul display.

ARDUINO – SENSORE REED:

L'ultima parte dello Sketch è la più difficile.

Per leggere la velocità in chilometri orari ci serve di sapere in quanto tempo la ruota della bicicletta compie un giro completo.

Per leggere i giri della ruota usiamo un sensore magnetico per allarmi che si mette sopra le porte o le finestre di casa e che è composto da un magnete e da un sensore sensibile al campo magnetico che si chiama REED.



Il sensore è composto da una capsula di vetro riempita di un gas inerte, isolante per la corrente elettrica. Questo gas impedisce l'ossidazione del contatto elettrico.

Quando il magnete si avvicina al sensore Reed la placchetta del contatto di materiale ferro magnetico viene attratta e si sposta in avanti chiudendo il contatto e permettendo il passaggio della corrente elettrica. Se fissiamo il magnete ai raggi della ruota della bicicletta e il sensore alla forcella con delle fascette di plastica possiamo sapere quando la ruota compie un giro.

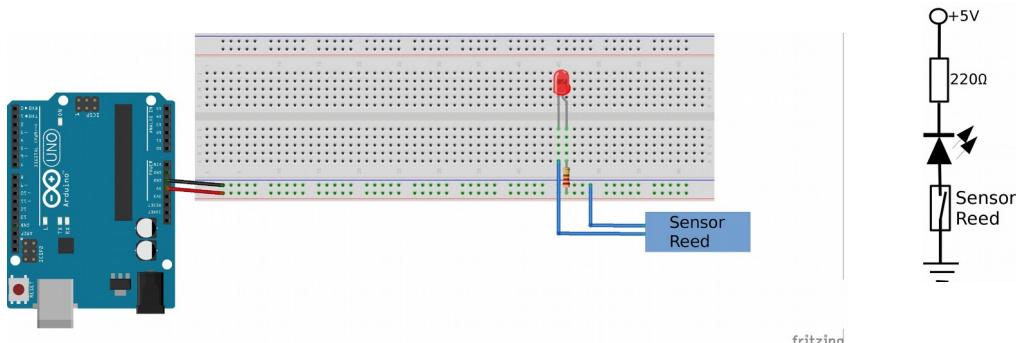
ARDUINO – L' INTERRUPT:



Possiamo fissare il magnete alla ruota legandolo ai raggi con una fascetta di plastica e il sensore alla forcella nello stesso modo facendo in modo che tra i due la distanza non sia più di un centimetro, nel caso possiamo montare un supporto per il sensore.

Per provare se il sensore legge il magnete lo collegiamo come in figura.

Sensore da porta



Facendo girare lentamente la ruota quando il magnete passa di fronte al sensore il LED si deve accendere. Se non si accende dobbiamo portare il sensore più vicino al magnete.

Ora che abbiamo il sensore operativo andiamo avanti a scrivere il codice.

Non possiamo leggere il sensore come siamo abituati a leggere un pulsante perché quando la ruota gira il sensore chiude il contatto per un tempo molto breve, giusto l'istante quando passa il magnete e se in quel momento Arduino sta scrivendo sul display o sta calcolando il tempo di viaggio non può vedere che l'ingresso del sensore ha cambiato di stato logico.

Quando abbiamo bisogno, come in questo caso, di una reazione immediata del microcontrollore dobbiamo usare l'INTERRUPT che significa "interrompere".

Arduino Uno ha due circuiti di Interrupt (interrupt 0 e interrupt 1) collegati alle connessioni 2 e 3. L'interrupt può vedere se un ingresso cambia di stato logico e fermare l'esecuzione del programma chiamando una funzione. Quando la funzione termina il programma riprende dal punto dove si era interrotto. In questo modo quando il sensore chiude il contatto la funzione che legge il sensore si attiva automaticamente.

ARDUINO – L' INTERRUPT:

Usare l'interrupt è molto semplice, l'istruzione si scrive nel "setup()".

```
attachInterrupt(numero del Interrupt, nome della funzione, modo);
```

Come numero di Interrupt mettiamo 0 se il sensore è collegato all'ingresso digitale 2 o mettiamo 1 se lo colleghiamo al 3.

Dopo la virgola scriviamo il nome della funzione che vogliamo chiamare e per ultimo scriviamo il modo.

L'interrupt ha quattro modi di lettura dell'ingresso:

LOW chiama la funzione quando l'ingresso ha un valore basso
CHANGE chiama la funzione quando l'ingresso cambia di stato
RISING chiama la funzione quando l'ingresso passa da basso a alto
FALLING chiama la funzione quando l'ingresso passa da alto a basso

Come si può vedere è molto flessibile ma ha delle limitazioni come la funzione "delay()" che non funziona in una funzione chiamata dall'interrupt.

L'interrupt si può disabilitare con l'istruzione:

```
detachInterrupt(numero del interrupt);
```

e si può riattivare con l'istruzione:

```
interrupts(numero del interrupt);
```

possiamo scrivere nel "setup()" del nostro programma:

```
attachInterrupt(1, Kmh, LOW); // attivo l' interrupt 1 che chiama la funzione "Kmh" quando l'ingresso 3 va basso.
```

Ora quando l'ingresso 3 si porta in stato logico basso l'interrupt chiama la funzione "void Kmh(){ " che adesso scriviamo.

PROGRAMMAZIONE – CICLO-COMPUTER:

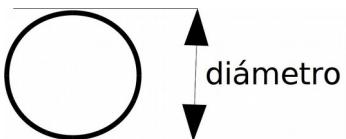
In questa funzione calcoliamo la velocità.

Il calcolo per conoscere i chilometri all'ora è:

Metri al secondo * 3.6

se con la bicicletta stiamo percorrendo 5 metri al secondo in un ora faremo 18 chilometri.

Per sapere quanta strada stiamo percorrendo ci serve conoscere il diametro della ruota in centimetri per calcolare la circonferenza.



Il diametro lo moltiplichiamo per 3.14 che è il valore del pi greco π .

Se abbiamo una ruota con un diametro di 65 cm avremo una circonferenza di 204.1 cm.

$$65 * 3.14 = 204.1$$

Però ci serve la circonferenza in metri quindi dividiamo il valore per 100 perché un metro è formato da 100 cm.

$$204.1 / 100 = 2.041$$

Tutte le volte che la ruota fa un giro la bicicletta percorrerà 2.041 metri di strada.

Se la ruota compie un giro al secondo stiamo percorrendo:

$$2.041 * 3.6 = 7.3476 \text{ chilometri all'ora.}$$

Nel nostro Sketch dobbiamo misurare quanto tempo impiega la ruota per fare un giro.

PROGRAMMAZIONE – CICLO-COMPUTER:

Per misurare il tempo di un giro-ruota usiamo la funzione "millis()". Quando l'ingresso va basso perché il magnete è di fronte al sensore memorizziamo il tempo in millisecondi del contatore "millis()" e quando si porta a basso una seconda volta memorizziamo un altro tempo. Se sottraiamo il tempo vecchio dal tempo nuovo otteniamo il tempo di un giro-ruota. Se dalla sottrazione ottengo 750 ms (millisecondi) devo fare il calcolo con la formula:

metri al secondo * 3.6

però in questa formula il tempo è una costante e quello che cambia è la distanza percorsa. Nel nostro caso la distanza è una costante perché è il diametro della ruota e quello che cambia è il tempo che impiega la ruota per fare un giro. Oltre tutto il nostro tempo è in millisecondi.

La formula in questo caso diventa;

```
velocita = ( circonferenza * 3600 ) / tempo_del_giro;
```

Moltiplichiamo per 3600 perché abbiamo un tempo in millisecondi.

Ora pare tutto pronto per scrivere la funzione però abbiamo ancora un problema da risolvere: nella funzione chiamata dall'Interrupt la funzione "delay()" non si può usare.

Perché è un problema?

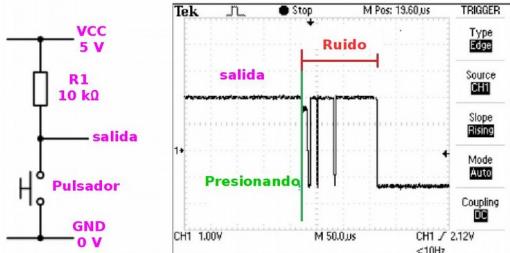
Perché quando collegiamo un pulsante ad Arduino dopo la lettura inseriamo sempre nel codice un piccolo "delay()" e lo facciamo perché il contatto meccanico come quello di un pulsante, di un sensore magnetico o uno switch di fine corsa non commuta in modo 'pulito' ma genera un 'rumore'.

Cos'è il rumore di un contatto elettrico?

La placchetta metallica che chiude il contatto elettrico ha una vibrazione meccanica che chiude e apre il contatto alcune volte prima di stabilizzarsi, oltre a questo il punto di contatto potrebbe avere ossidazione (l'ossido è isolante) o righature da usura. Questo aprire e chiudere il contatto si chiama 'rumore'.

Siccome Arduino è rapidissimo il 'rumore' per lui è un ingresso che cambia di stato più volte.

PROGRAMMAZIONE – CICLO-COMPUTER:



Nella figura si vede nel monitor di un oscilloscopio il 'rumore' di un pulsante.

Quindi dobbiamo scrivere un controllo che se abbiamo una chiamata dal Interrupt in meno di 76 millisecondi dalla precedente non la considereremo valida.

Scriviamo la funzione che chiamerà l'Interrupt e la chiamiamo:

```
void Kmh(){
```

```

84
85 void Kmh(){ // calcolo la velocità
86
87 x_nuovo=millis(); // in x nuovo memorizzo il valore di millis()
88 if (x_nuovo>x_vecchio+76){ // se il valore nuovo è minore del valore vecchio più 76 milli secondi è un rumore
89 // del sensore e non è valida la lettura, se è maggiore la lettura è valida
90 tempo_del_giro=x_nuovo-x_vecchio; // il tempo del giro della ruota è la differenza del tempo nuovo-il vecchio
91 x_vecchio=x_nuovo; // memorizzo il valore nuovo nel vecchio per prendere un nuovo valore
92 }
93
94 velocita=(circonferenza*3600)/tempo_del_giro;// calcolo la velocità
95 c=1; // metto a 1 la variabile di controllo
96
97 }
```

In linea 87 mettiamo nella variabile “x_nuovo” il valore di “millis()” che è il nuovo valore del tempo.

In linea 88 controlliamo se la chiamata della funzione è vera o è per il rumore del contatto. Se il nuovo tempo è maggiore di 76 ms rispetto al tempo vecchio la chiamata è vera quindi

In linea 90 calcoliamo il tempo del giro della ruota.

In linea 91 mettiamo il valore di “x_nuovo” nella variabile “x_vecchio” perché ora è un dato vecchio.

In linea 94 calcoliamo la velocità che scriverà il display.

In linea 95 mettiamo a 1 la variabile “c” che è la variabile di controllo che ci serve per sapere se la ruota è ferma perché se la ruota si ferma l'Interrupt non chiamerà la funzione e sul display resterà l'ultimo valore calcolato.

PROGRAMMAZIONE – CICLO-COMPUTER:

Nel “loop()” scriviamo la parte di codice per mettere a zero la velocità se si ferma la ruota.

```
60 | tempo_di_viaggio();           // chiamo la funzione "tempo di viaggio"
61 | temperatura();              // chiamo la funzione per leggere la temperatura
62 | if (c==1) c1=millis();        // controllo se la ruota è ferma memorizzando il valore di millis()
63 | if (c==0){                  // se la variabile di controllo è 0 è perchè la funzione Kmh non si è eseguita
64 |   if((c1+2000)<millis()) velocita=0; // se dopo due secondi non si esegue la funzione kmh è perché
65 |   // la ruota è ferma e metto a 0 la velocità
66 |   c=0;                      // metto a 0 la variabile di controllo
67 |
68 --
```

In linea 62 se la variabile di controllo “c” è uguale a 1 vuol dire che il Interrupt ha chiamato la funzione. Mettiamo nella variabile “c1” il valore di “millis()” per sapere quando si eseguì l’ultima chiamata del Interrupt.

In linea 63 controlliamo se la variabile di controllo è uguale a zero. Se è uguale a zero controlliamo da quanto tempo non si esegue la chiamata del Interrupt.

In linea 64 se il tempo della variabile “c1” più 2000 è minore del valore di “millis()” significa che il Interrupt non sta chiamando da 2 secondi quindi la ruota è ferma. Metto a zero la variabile “velocita”.

Ora dobbiamo creare tutte le nuove variabili dalla linea 20 alla 26.

Tutte le variabili che memorizziamo il valore della funzione “millis()” saranno di tipo “unsigned long”.

Il sensore lo collegiamo alla connessione 3.

Nel setup aggiungiamo la linea:

```
pinMode ( sensore, INPUT_PULLUP );
```

In modo che Arduino inserisca la resistenza di pull-up all'ingresso e si possa collegare il sensore in modo diretto e poi la linea che attiva il Iinterrupt:

```
attachInterrupt ( 1, Kmh, LOW );
```

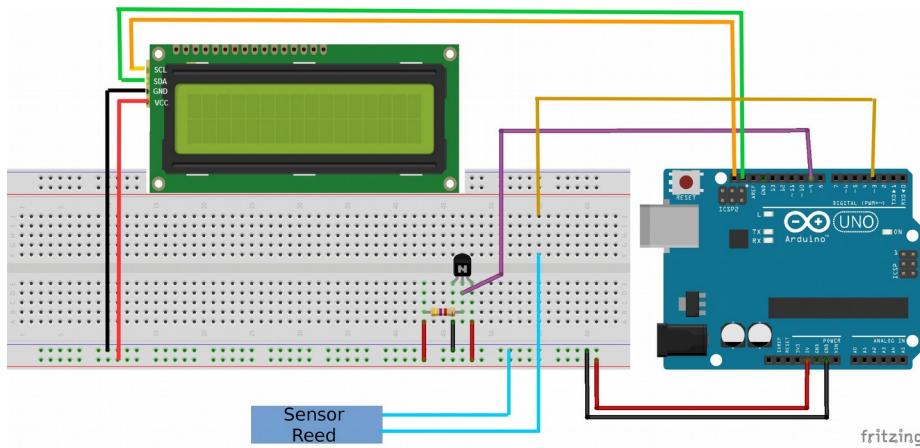
PROGRAMMAZIONE – CICLO-COMPUTER:

```

12 byte velocita=0; // memorizzo la velocità in kmh
13 float gradi=0; // memorizzo la lettura della temperatura
14 byte ore=0; // memorizzo le ore di viaggio
15 byte minuti=0; // memorizzo i minuti di viaggio
16 byte secondi =0; // memorizzo i secondi di viaggio
17 long tempo =0; // memorizza in secondi il tempo di viaggio
18 float tmp =0; // è una variabile temporale per il calcolo delle ore, minuti, secondi
19
20 #define sensore 3 // dove collego il sensore reed
21 unsigned long x_nuovo=0; // memorizza il valore di millis() quando passa il magnete davanti al sensore
22 unsigned long x_vecchio=0; // memorizza il tempo della lettura precedente
23 int tempo_del_giro=0; // memorizza il tempo di un giro della ruota
24 float circonferenza=1.97; // diametro in metri della ruota
25 byte c=0; // è la variabile di controllo se si ferma la ruota
26 unsigned long cl=0; // memorizza il valore di millis() per vedere se la ruota è ferma
27
28 #define DISPLAY 5 // è il collegamento del led del display
29 #define foto_resistenza 0 // è il collegamento della fotoresistenza
30
31 void setup()
32 {
33   lcd.init(); // inizializzo il display
34   lcd.backlight(); // accendo il led del display
35   sensorDS18B20.begin(); //inizializzo la libreria del sensore di temperatura
36   pinMode(sensore,INPUT_PULLUP); // imposto come ingresso il pin del sensore reed
37   attachInterrupt(1, Kmh, LOW); // attivo l'interrupt che chiama la funzione Kmh() quando l'ingresso 3 va basso
38
39 }

```

Il sensore lo collegiamo alla connessione 3 di Arduino e al GND della basetta di prova.

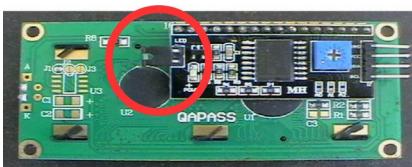


PROGRAMMAZIONE – CICLO-COMPUTER:

Come ultima cosa dobbiamo collegare la foto resistenza per regolare la luminosità del display.

Non è possibile regolare la luminosità del display con un comando software perché la scheda di conversione I2C-parallelo non genera il segnale PWM necessario per la regolazione.

Se osserviamo la parte posteriore del display noteremo sulla scheda dove abbiamo collegato il bus I2C che sul lato opposto si trova un piccolo connettore come si vede in figura.



1



2



3

In figura 1 si vede evidenziato il connettore del LED chiuso da un ponticello (Jumper) se lo sfiliamo (figura 2) il LED si spegne. Osservando il connettore troviamo la scritta “Led” sulla connessione dal lato del collegamento col display che collega l'anodo del LED. L'altro polo è collegato al 5V.

Il terminale con la scritta “Led” lo collegiamo (figura 3) all'uscita analogica di Arduino per modificare la luminosità del display. Se sul vostro display non trovate la scritta “Led” facciamo riferimento alla posizione dei terminali dal momento che queste schede sono industrializzate tutte uguali ma per non sbagliare, per sicurezza mettiamo un diodo in serie al collegamento.

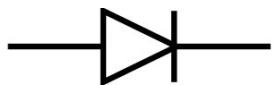
È pericoloso collegare una tensione ad una uscita digitale perché se l'uscita si trova in stato logico alto non è un problema ma se va in stato logico basso circola una corrente sufficiente a bruciare il micro controllore.

Mettendo un diodo in serie all'uscita di Arduino non accade nulla se sbagliamo a collegare il terminale del LED perché il diodo impedisce il ritorno della corrente inversa.

I DIODI:

I diodi sono componenti elettronici che permettono il flusso della corrente elettrica in una sola direzione. Si comportano come una valvola idraulica di non ritorno che permette all'acqua di circolare in una direzione ma se l'acqua prova a tornare indietro si chiude.

I terminali del diodo si chiamano Anodo (+) e Catodo (-).



I diodi al silicio sono rappresentati con il simbolo in figura.



Sono di forma cilindrica e hanno un anello da un lato che identifica il terminale del Catodo.



Nel diodo la corrente può circolare solo in un senso e grazie a questa caratteristica si usano per raddrizzare una corrente alternata per bloccare un segnale o per dividere un segnale nelle sue componenti positive e negative. I diodi si differenziano per corrente, voltaggio e velocità operativa oltre che per caduta di tensione.

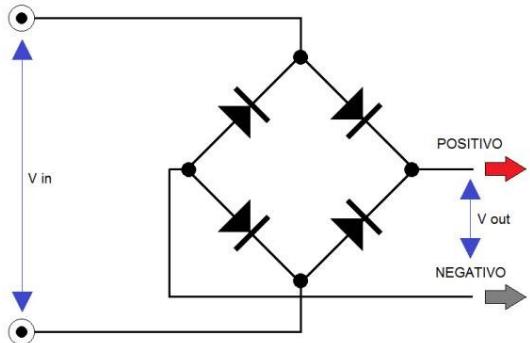
Un diodo è composto da due materiali uno di tipo "P" e uno di tipo "N" come il transistor.

Le lacune generate da questa operazione fanno sì che si generi un semiconduttore ovvero un materiale che può condurre corrente elettrica solo in determinate condizioni.

Se all'Anodo colleghiamo il positivo dell'alimentazione e al Catodo il negativo il diodo si comporta come un interruttore chiuso permettendo il flusso della corrente. Se invertiamo la polarità si comporta come un interruttore aperto bloccando il passaggio della corrente elettrica.

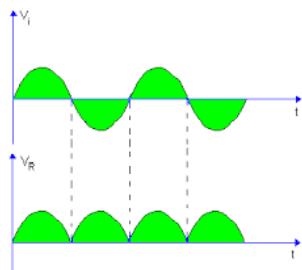
I diodi quando conducono provocano sempre una caduta di tensione. Nel caso del diodo al silicio la caduta di tensione è di circa 0.6 – 0.7 Volt.

I DIODI:



Per raddrizzare una corrente alternata, ovvero trasformarla in corrente continua si usano quattro diodi in configurazione 'a ponte' detta anche 'ponte di diodi' come si vede in figura.

Per trasformare la corrente alternata in continua i diodi che si usano sono quelli al silicio che pertanto provocano una caduta di tensione di 0.6 – 0.7 Volt.



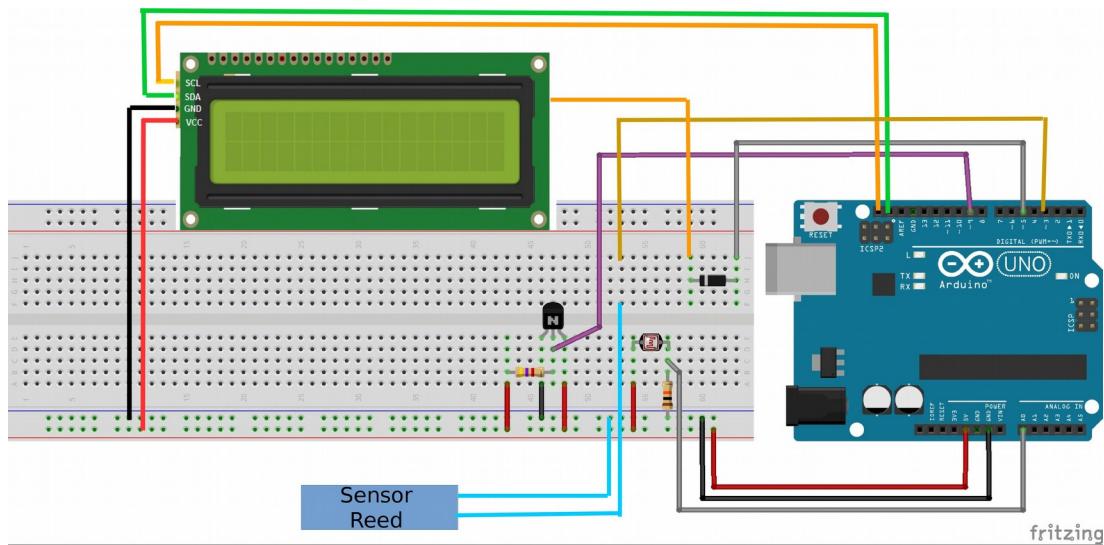
La corrente alternata si divide nel ponte raddrizzatore seguendo il senso dei diodi e in uscita troveremo una corrente polarizzata ma pursempre pulsante come si vede in figura. Per renderla continua si mettono in parallelo all'uscita del ponte di diodi dei condensatori elettrolitici in grado di stabilizzare le oscillazioni di corrente rendendola continua.



I ponti di diodi sono venduti già incapsulati in contenitori plastici di svariate forme e identificati dalla sigla che ne definisce il contenitore (Case).

PROGRAMMAZIONE – CICLO-COMPUTER:

Se per caso ci dovessimo collegare al terminale 5V del display quando l'uscita di Arduino va bassa il diodo impedisce alla tensione positiva di raggiungere l'uscita. Ora è chiaro perché mettendo un diodo in serie all'uscita di Arduino non rischiamo di danneggiare la scheda elettronica.



Collegiamo il display e la foto resistenza come da schema. Adesso scriviamo nel "loop()" il codice per modificare la luminosità del display.

```

71 | if ((analogRead(foto_resistenza)/4) < 60) analogWrite(DISPLAY,60); // metto un limite alla luce del display
72 | if ((analogRead(foto_resistenza)/4) > 60) analogWrite(DISPLAY, (analogRead(foto_resistenza)/4));
73 |
74 |

```

In linea 71 metto un limite alla luminosità in modo che non si spenga se non c'è luce.

In linea 72 scrivo l'uscita analogica PWM con il valore della foto resistenza diviso 4.

PROGRAMMAZIONE – CICLO-COMPUTER:

Aggiungiamo nelle dichiarazioni dove abbiamo collegato il LED del display e la foto resistenza.

```
28 #define DISPLAY 5      // è il collegamento del led del display  
29 #define foto_resistenza 0    // è il collegamento della fotoresistenza  
30
```

Lo Sketch è finito. Possiamo farci una bella pedalata.

Finora avete scoperto molte cose del fantastico mondo di Arduino.

Avete appreso che nulla è fine a se stesso, come un programma complesso sia formato da programmi semplici e come sia importante il pensiero logico e creativo per risolvere i problemi di programmazione. La matematica ha un ruolo fondamentale perché le macchine possono solo 'pensare' in modo matematico e con la matematica abbiamo risolto il calcolo del tempo e dei chilometri orari.

Avete appreso molte istruzioni che vi permetteranno di fare tante cose interessanti e molte ancora non le conoscete ma si imparano col tempo e con la pratica.

Nell'ultima parte di questo primo volume scopriremo come si interfaccia Arduino al mondo di potenza.

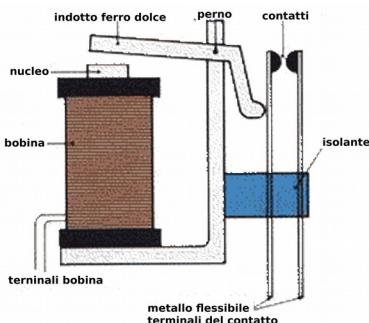
Il micro controllore di Arduino non ha corrente sufficiente per comandare in modo diretto motori o altri utilizzatori con tensioni differenti e con molto assorbimento di corrente.

Questa non è una limitazione.

Come prima cosa scopriremo come collegare un relè per collegare dei motori o altri utilizzatori.

Il relè è un componente elettrico che ci permette di collegare apparati che funzionano anche in corrente alternata come lampade e motori a 220V CA senza distruggere il micro controllore.

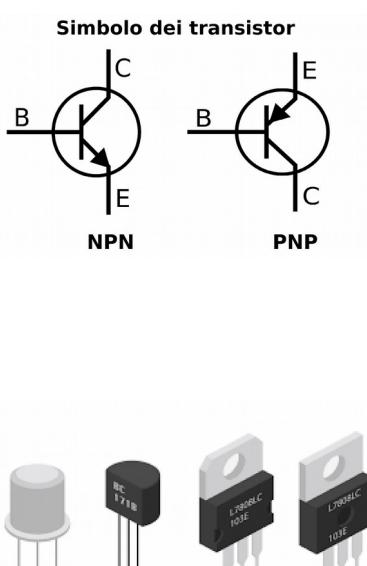
COLLEGARE UN RELE E UN TRANSISTOR:



Il relè è un componente elettrico che funziona come un interruttore, apre e chiude un circuito azionato dalla corrente elettrica. Il relè apre e chiude un circuito azionato da una calamita.

Quando alimentiamo la bobina questa crea un campo magnetico (elettro magnete) attirando un leveraggio che spinge il contatto di potenza chiudendolo.

Il modo più semplice per comandare un relè con un circuito di controllo elettronico è usare un transistor NPN.



Il transistor si usa in elettronica per amplificare qualunque tipo di segnale elettrico. Il simbolo del transistor è un cerchio con tre uscite E, B, C.

La lettera E indica l'emettitore, la lettera B indica la base e la lettera C indica il collettore.

Osservando il simbolo del transistor si può notare che la freccia identifica il tipo PNP e il tipo NPN.

Nel transistor PNP il collettore si collega al negativo (GND) e nel NPN al positivo (VDD). In caso di dubbi è sufficiente riferirsi alla lettera centrale: PNP N=negativo NPN P=positivo.

La base del transistor si polarizza sempre in modo inverso. Il transistor è un componente che lavora in corrente quindi si pone in conduzione tra il collettore e l'emettitore se una corrente polarizza la sua base.

La base del transistor è molto sensibile e quindi è sufficiente una piccola corrente per portarlo in conduzione. Se lo si polarizza con una corrente inversa il transistor blocca il passaggio della corrente portandosi in 'interdizione'.

COLLEGARE UN RELE E UN TRANSISTOR:

Collegando un transistor NPN a una uscita logica di Arduino vedremo il transistor andare in conduzione quando l'uscita si porta alta (+ 5V) facendo passare la corrente negativa tra il collettore e l'emettitore e andare in interdizione quando l'uscita logica passa a bassa (GND) bloccando il passaggio della corrente. Quindi il transistor non solo inverte la logica dell'uscita (con l'uscita alta passa il GND) ma quando non conduce non passa nulla quindi all'uscita non avremo più un segnale digitale.

I transistor si differenziano per corrente, voltaggio, velocità operativa e forma.

Sui manuali tecnici (datasheet) possiamo trovare le caratteristiche dei transistor.

VCB = 45 V Max

VCE = 30 V Max

VEB = 6 V Max

IC = 100 mA

Ptot = 300mW

Hfe = 100-200

Ft = 50 MHz

VCB indica che questo transistor può sopportare una tensione massima tra collettore e base di 45 Volt.

VCE indica che la tensione massima che si può collegare tra il collettore e l'emettitore non potrà essere uguale o superiore a 30 Volt. Quindi lo possiamo usare per progetti alimentati al massimo a 24 Volt.

VEB indica che la massima tensione inversa sopportata non deve essere maggiore di 6 Volt.

IC indica la corrente massima che può passare nel collettore per un breve momento.

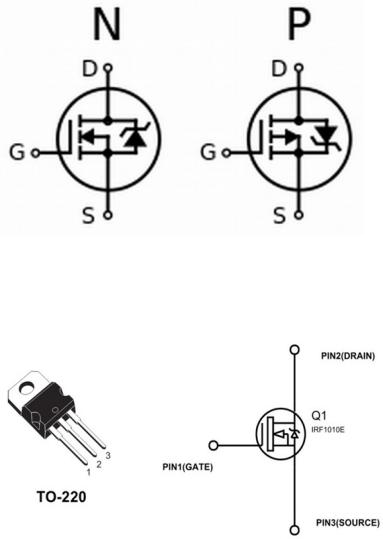
Ptot indica la potenza massima che il transistor può dissipare alla temperatura di 25 gradi Celsius.

Hfe indica che il transistor può amplificare da 100 a 200 volte.

Ft indica la massima frequenza di lavoro.

I transistor hanno una resistenza interna abbastanza alta e una corrente di lavoro limitata, per pilotare motori o altri componenti energivori è meglio usare i MOSFET.

IL MOSFET:



MOSFET significa (Metal Oxide Semiconductor Field Effect) semiconduttore all'ossido di metallo ad effetto di campo.

Come i transistor si usano per amplificare un segnale però presentano una resistenza interna molto bassa e una corrente di lavoro molto alta. A differenza del transistor si comanda in tensione.

Come i transistor hanno le loro caratteristiche e si differenziano per essere di canale P o N sempre indicato nello schema dal senso della freccia.

I terminali sono identificati dalle lettere D,G,S dove D indica il Drain, G indica il Gate e S indica il Source.

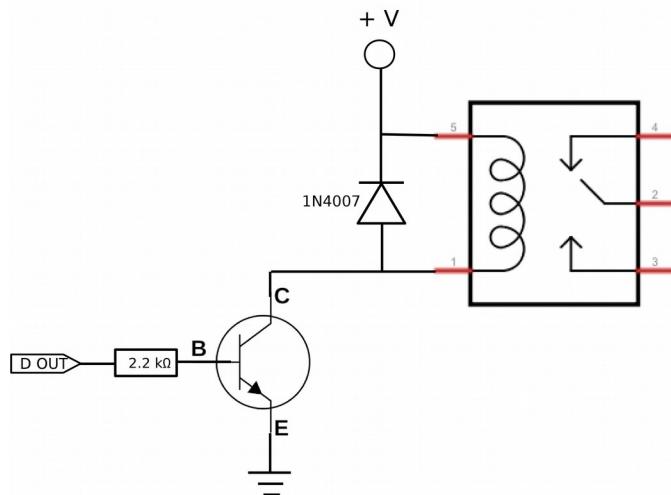
I MOSFET lavorano in tensione e dal manuale si ricava la tensione di polarizzazione del Gate. Se collego un MOSFET tipo IRF520 che ha una tensione di polarizzazione di 10 Volt e una corrente di lavoro di 9.2 Amper e lo collego ad Arduino che ha una uscita di 5 Volt, il massimo passaggio di corrente che otterrò tra il Drain e il Source sarà di 2 Amper.

Esistono MOSFET di tipo "Logic Level" livello logico che lavorano con 5 Volt specifici nel nostro caso che sono: IRLZ44N, FQP30N06L y STP36NE06 dove il primo è il più comune. Con questi MOSFET possiamo usare tutta la corrente che possono fornire.

Collegiamo Arduino ad un relè e a un motore.

COLLEGARE UN RELE E UN TRANSISTOR:

Per collegare un relè ad Arduino ci serve un transistor, un diodo e una resistenza.



All'uscita digitale di Arduino collegiamo la resistenza che abbassa la corrente per non bruciare la base del transistor. Possiamo usare un transistor di tipo BC337 che è economico o un equivalente.

L'emettitore del transistor lo collegiamo al GND di Arduino e il collettore al relè. In parallelo alla bobina del relè mettiamo un diodo perché la bobina genera delle correnti che se raggiungono il transistor lo bruciano. Con il diodo quando si generano queste correnti soprattutto quando si interrompe il flusso magnetico si scaricano nel diodo senza raggiungere il transistor.

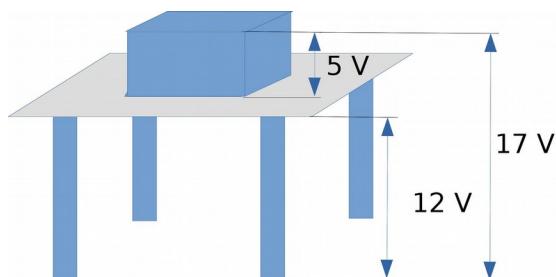
L'altro capo della bobina lo collegiamo al 5 Volt se abbiamo un relè a 5 V o al positivo del 12 Volt se abbiamo un relè a 12 Volt. In questo caso dobbiamo usare un alimentatore supplementare da 12 Volt collegato con il GND al GND di Arduino.

Quando usiamo una fonte di alimentazione secondaria il GND dell'alimentatore secondario va sempre collegato al GND di Arduino diversamente il circuito non funziona o si può danneggiare.

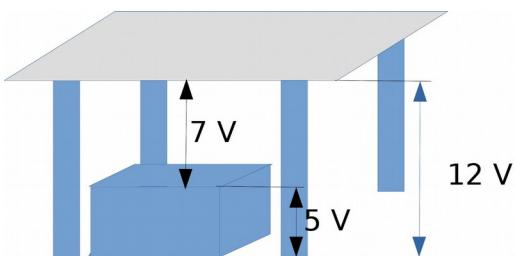
Tutti i collegamenti di GND devono essere uniti ed equipotenziali. Vediamo di capirlo meglio con un esempio.

Se appoggiamo una scatola su un tavolo e vogliamo sapere quanto è alta la scatola e quanto è alto il tavolo misuriamo la scatola dalla base al coperchio e il tavolo dal piano al pavimento ottenendo due misure. Per misurare la tensione di Arduino la misuro dal +5V al GND e misuro 5 Volt. Per misurare la tensione del secondo alimentatore misuro tra il +12V e il GND e ottengo 12 Volt.

COLLEGARE UN RELE E UN TRANSISTOR:

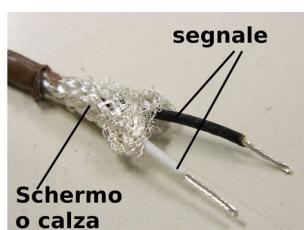


In questo caso ottengo dei valori relativi perché il valore assoluto si misura dal pavimento, dalla terra, ground. Non è un caso se il zero Volt lo si è chiamato terra quindi se realmente voglio la misura reale e assoluta dei due oggetti li devo appoggiare entrambi al pavimento. Oltretutto in questo esempio ho appoggiato la scatola sul tavolo ma potrebbe trovarsi ovunque nello spazio. Mettendo la scatola sul pavimento i due oggetti hanno lo stesso punto di riferimento.



Non è un caso se il collegamento di GND è sempre il primo che si fa.

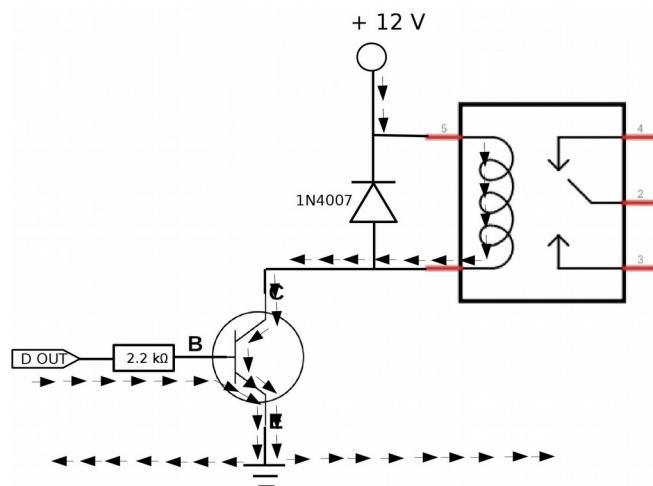
Con questo esempio spero sia chiaro come si misurano le differenze di potenziale elettrico in un circuito e tra due alimentazioni in un circuito elettrico e elettronico. In elettronica tutto è collegato sia fisicamente che teoricamente. Avevamo già visto la differenza di potenziale elettrico nel partitore di tensione e con questo esempio ora sappiamo anche il motivo e l'importanza che hanno i collegamenti equipotenziali di GND.



I segnali analogici con tensioni molto basse come i segnali audio, video analogico e i segnali di sensori analogici che colleghiamo all'ingresso analogico di Arduino risentono dell'induzione elettromagnetica che c'è nell'aria e per questo si collegano con cavi 'schermati' che hanno una calza che si collega al GND. In questo modo tutti i disturbi captati dallo schermo sono scaricati a GND e non si concatenano al segnale analogico.

COLLEGARE UN RELE E UN TRANSISTOR:

Come scrisse si possono usare relè con tensioni differenti collegando il GND dell'alimentazione secondaria al GND di Arduino perché le due correnti non possono mischiarsi grazie al transistor come si può vedere in figura.



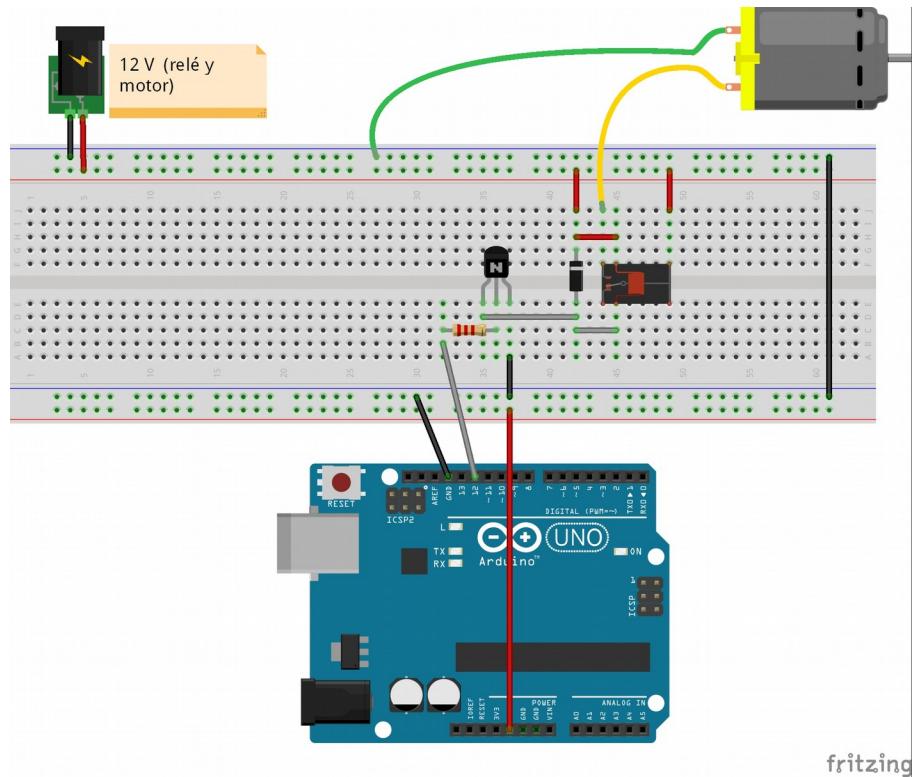
Il relè è collegato in modo diretto al 12 Volt dell'alimentazione secondaria e il transistor interrompe il GND perché è di tipo NPN e si porta in conduzione con una corrente positiva.

Se carichiamo in Arduino il programma che abbiamo scritto per far lampeggiare il LED e colleghiamo il transistor potremo sentire il ticchettio del relè quando apre e quando chiude il contatto.

Guardando lo schema di montaggio possiamo notare che ora la basetta di prova è divisa in due parti una a 12 Volt e una a 5 Volt. Le due alimentazioni hanno il GND collegato insieme.

Possiamo collegare un motore a 12 Volt in corrente continua e quando il relè chiuderà il contatto il motore girerà.

COLLEGARE UN RELE E UN TRANSISTOR:



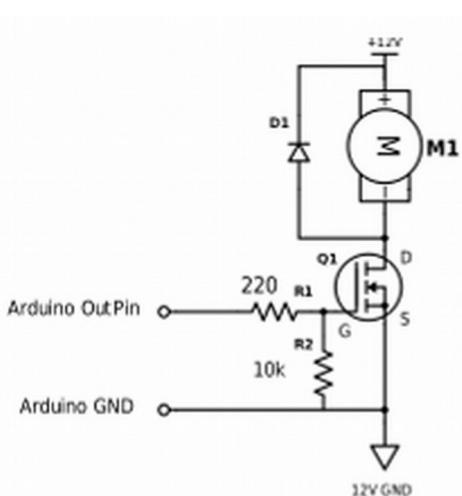
fritzing

I relè hanno collegamenti differenti, prima di collegare il relè occorre verificare il suo schema (pinout) per essere sicuri dove si trovino i terminali della bobina e il contatto NA.

Se usiamo un MOSFET non ci serve il relè per pilotare il motore in corrente continua perché il MOSFET è un componente di potenza e lo può alimentare direttamente.

Scopriamo come si usa questo componente.

ARDUINO - COLLEGARE UN MOSFET:



Nello schema elettrico possiamo vedere come si collega un MOSFET che pilota un motore in corrente continua a 12 Volt.

Lo schema somiglia a quello del transistor che comanda il relè.

La resistenza R1 da $220\ \Omega$ non sarebbe necessaria perché il MOSFET si comanda in tensione ma la mettiamo per proteggere l'uscita di Arduino.

Anche la resistenza R2 di pull-down non sarebbe necessaria perché Arduino ha una uscita digitale che quando è bassa manda il MOSFET in interdizione.

Però se togliamo l'alimentazione ad Arduino senza scollegare l'alimentazione secondaria a 12 Volt sull'uscita di Arduino non ci saranno più i 5 Volt ma neppure i 0 Volt del GND e il Gate del MOSFET riceverà del rumore elettromagnetico che potrebbe portarlo in conduzione avviando il motore in modo imprevisto. Con la resitenza R2 impediamo che il motore possa muoversi se viene a mancare l'uscita di Arduino.

Come nello schema del transistor e del relè mettiamo un diodo in parallelo al motore per scaricare le correnti prodotte dal motore durante la marcia e quando si ferma perché un motore elettrico in corrente continua si comporta anche da dinamo e le correnti che autoproduce potrebbero danneggiare il MOSFET.

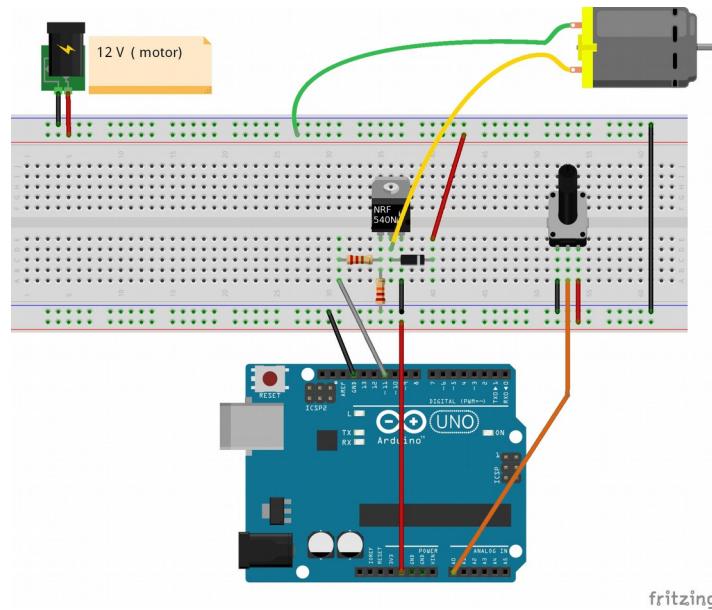
Possiamo caricare in Arduino lo Sketch che avevamo scritto per regolare la luminosità del LED con il potenziometro per regolare la velocità del motore.

In questo caso colleghiamo il MOSFET che comanda il motore all'uscita analogica 11.

ARDUINO - COLLEGARE UN MOSFET:

```
1 #define MOTORE 11
2 #define POTENZIOMETRO 0
3
4 int valore = 0; // in "valore" metto la lettura del potenziometro
5
6 void setup(){
7   pinMode ( MOTORE, OUTPUT );
8 }
9
10
11 void loop(){
12   valore = analogRead ( POTENZIOMETRO ); // in "valore" metto la lettura del potenziometro
13   valore = valore / 4; // divido la variabile
14   analogWrite ( MOTORE, valore ); // scrivo la velocità del motore
15 }
16
```

Modifichiamo il programma come in figura e collegiamo i componenti come nello schema di cablaggio sulla basetta di prova.



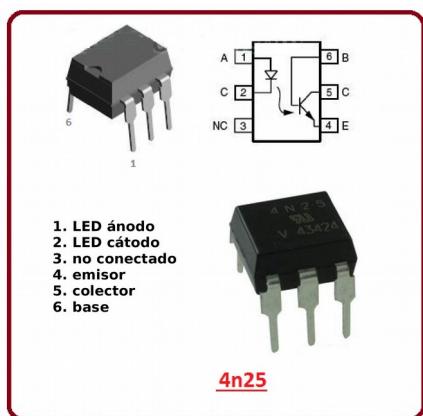
Carichiamo il programma in Arduino e girando l'alberino del potenziometro varierà la velocità del motore.

IL FOTO ACCOPPIATORE:

Collegare un componente di potenza come un motore elettrico in corrente continua o delle strip LED con lo schema che abbiamo provato potrebbe essere controproducente perché il motore come le strip LED generano molte correnti durante il loro funzionamento che risalendo dal collegamento del GND vanno a disturbare il micro controllore portandolo ad avere comportamenti imprevisti e non gestibili.

In tutti i circuiti professionali il micro controllore è schermato e protetto dal mondo esterno e non condivide nulla. Per collegare Arduino separandolo completamente dal mondo esterno si usano componenti elettronici che si chiamano “foto accoppiatori”.

Un foto accoppiatore è un componente che dentro il suo corpo (case) ha un LED e un foto transistor. Un foto transistor è simile ad una foto resistenza, si pone in conduzione quando la base è colpita da una energia fotonica, una luce.

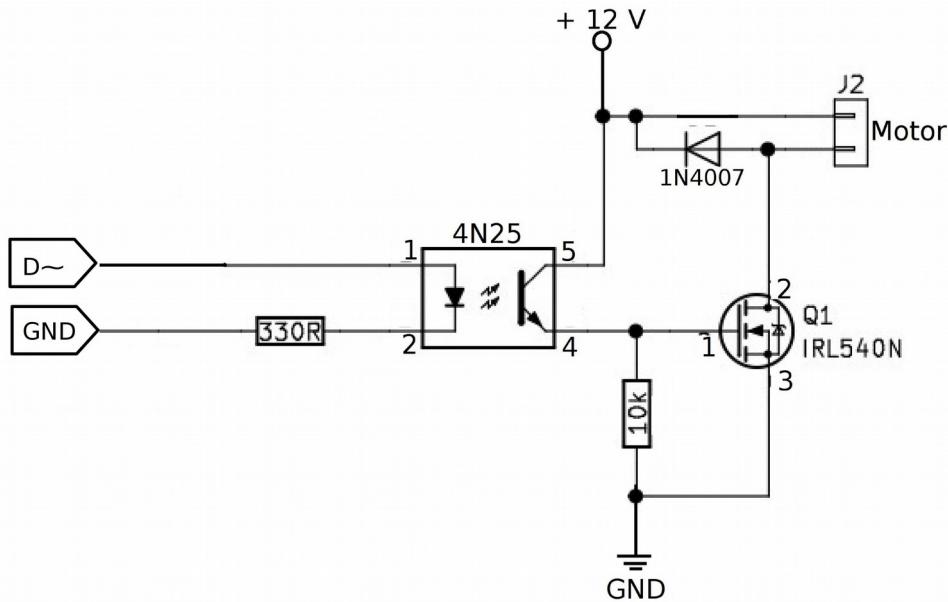


Quando i fotoni colpiscono la base foto sensibile del transistor lo portano in conduzione.

In questo modo è possibile comandare il transistor senza collegare il suo GND in comune con quello di Arduino in modo che nulla possa infastidire o danneggiare il micro controllore.

In figura si vede il modello 4N25 che ha solo un fotoaccoppiatore, esistono modelli che hanno 2 o 4 fotoaccoppiatori e si distinguono per l'ultimo numero della sigla 4N25-2, 4N25-4.

IL FOTO ACCOPPIATORE:



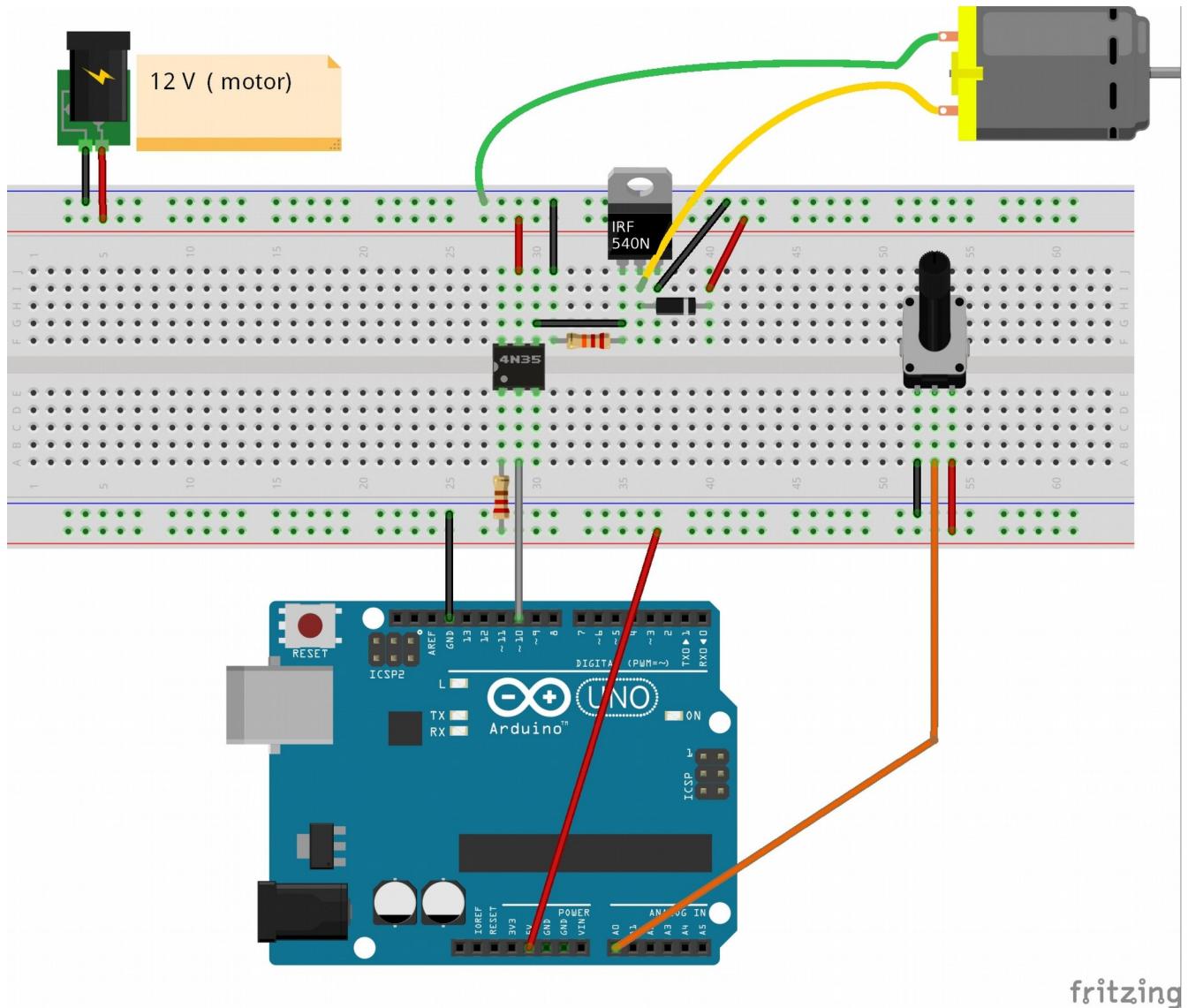
Nello schema elettrico si può vedere come adesso i due circuiti siano realmente separati.

Il mondo di Arduino finisce nel LED del foto accoppiatore e dall'altro lato inizia il mondo di potenza del motore.

Quando il LED si accende il transistor conduce permettendo alla corrente positiva del 12 Volt di raggiungere il Gate del MOSFET attivando la conduzione tra il Sorce e il Drain. Quando il LED si spegne il transistor va in interdizione interrompendo il positivo del 12 Volt. Il negativo del 12 Volt risale dalla resistenza da 10 KΩ raggiungendo il Gate del MOSFET portandolo in interdizione.

ARDUINO - COLLEGARE UN FOTO ACCOPPIATORE:

Collegiamo i componenti come nello schema di montaggio.



fritzing

Come si può notare la basetta di prova ora è divisa in due parti indipendenti con due alimentazioni diverse unite solo dal fascio luminoso di un LED.

IL PWM SEGUE:

Ora che abbiamo ben compreso come funziona l'uscita analogica di Arduino e come si usa voglio finire di parlarvi del PWM.

Arduino Uno ha tre oscillatori che generano la frequenza del PWM, il primo collegato alle uscite 5 e 6, il secondo alle uscite 9 e 10 e il terzo collegato alle uscite 11 e 3.

A tutti gli oscillatori sono collegati due comparatori che sono (OUTPUT COMPARE – OC) OCxA, OCxB. Così da ottenere OC0A, OC0B, OC1A, OC1B, OC2A y OC2B come si vede in figura.

| Timer | Frequenza di base | Comparatore di uscita | Uscita di Arduino | Frequenza automatica |
|-------|-------------------|-----------------------|-------------------|----------------------|
| TCCR0 | 62,500 Hz | OC0 | OC0A | 6 976 Hz |
| | 62,500 Hz | | OC0B | 5 976 Hz |
| TCCR1 | 31,250 Hz | OC1 | OC1A | 9 488 Hz |
| | 31,250 Hz | | OC1B | 10 488 Hz |
| TCCR2 | 31,250 Hz | OC2 | OC2A | 11 488 Hz |
| | 31,250 Hz | | OC2B | 3 488 Hz |

Nell'ultimo progetto che abbiamo realizzato si può provare a cambiare il collegamento del fotoaccoppiatore sulle uscite 5 o 6 per vedere come si comporta il motore con una frequenza maggiore. Sarà sufficiente cambiare il numero nel #define MOTORE.

Esiste la possibilità di cambiare la frequenza del PWM scrivendo una linea di codice nel "setup()":

```
setPwmFrequency(11,8);
```

dove il 11 è il numero dell'uscita e 8 è il divisore della frequenza di base. Impostando in questo modo otterremo sull'uscita 10 una frequenza di PWM di

$$31250 / 8 = 3906 \text{ Hz}$$

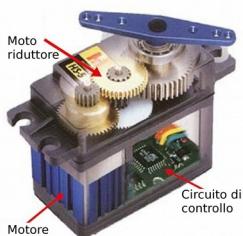
Il timer zero è usato dalla funzione "millis()" e "delay()" oltre che da alcune librerie come la "Servo" ed è meglio non giocarci.

Si può provare a cambiare la frequenza della connessione 11 per fare le prove.

IL SERVO MOTORE:

Ho parlato molto del PWM perché esiste un motore speciale che si chiama “servomotore”. Il servomotore è il primo che si usa per fare robotica e si gestisce con il PWM e con la classe “Servo”.

Un servomotore è un tipo speciale di motore che permette di controllare la posizione del suo asse. È progettato per muoversi un certo numero di gradi e rimanere fermo una volta raggiunta la posizione.



Al suo interno troviamo un comune motore in corrente continua che ha il suo asse collegato ad un motoriduttore che ne aumenta la forza di torsione e quella di frenatura per rimanere in posizione. Un circuito elettronico legge la posizione tramite un potenziometro collegato al suo asse e si incarica di portarlo nella posizione richiesta.

Un servo motore è un organo meccanico complesso che si distingue per forza di torsione, tensione operativa e corrente assorbita. Può avere ingranaggi di plastica o metallo a seconda dei modelli.

I servo motori hanno tre fili di collegamento. Generalmente i colori dei fili sono standarizzati e quindi facilmente riconoscibili.

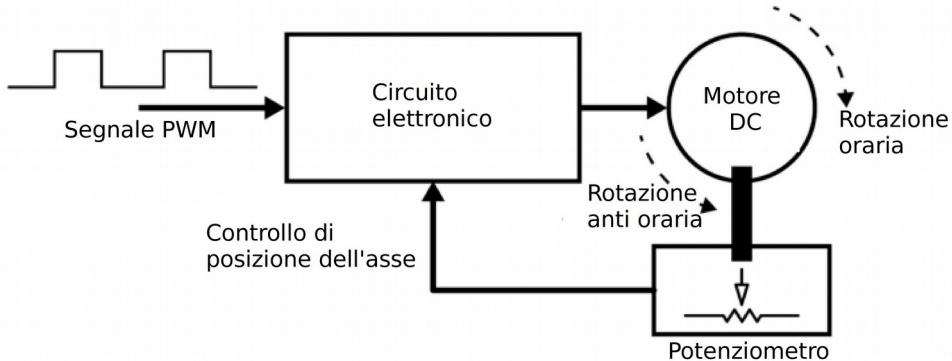
| Positivo di alimentazione | Negativo di alimentazione | Segnale di controllo |
|---------------------------|---------------------------|----------------------|
| Red | Black | Yellow |

I colori possono cambiare da un costruttore ad un altro. La necessità di un segnale di controllo fa sì che il servomotore non si possa usarlo senza un circuito adeguato.

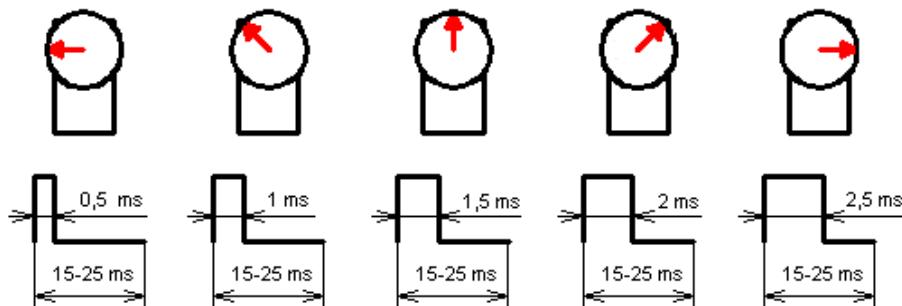
Per funzionare necessita di un segnale modulato PWM.

IL SERVO MOTORE:

DIAGRAMMA A BLOCCHI DI UN SERVO MOTORE



Il diagrammi a blocchi del servo motore rappresenta in modo visuale il servo motore come un sistema complesso. Il circuito elettronico è incaricato di ricevere il segnale PWM e trasformarlo in una posizione in gradi dell'asse del motore. Il potenziometro collegato all'asse del servo in modalità partitore resistivo comunica la posizione raggiunta.



Come si può vedere in figura il tempo della semionda positiva del PWM posiziona l'asse del motore in una posizione compresa tra 0 e 180 gradi.

Il modo migliore per gestire un servo motore è usare la classe "Servo".

PROGRAMMAZIONE - IL SERVO MOTORE:

La classe “Servo” si trova tra le librerie del IDE e si gestisce come una libreria qualunque.

Questa classe aggiunge alcune funzioni che ci facilitano la vita:

attach() nella parentesi scriviamo dove abbiamo collegato il servo motore.

attached() controlla che un servo motore sia collagato

detach() scollega un servo motore

read() legge la posizione dell'asse scritta con il “write”

write() scrive la posizione del servo motore

writeMicrosecond() è la velocità del servo motore da 1000 a 2000

La classe servo può usare tutte le uscite di Arduino e blocca l'istruzione “analogWrite”.

Scriviamo uno Sketch molto semplice per provare il servo motore facendo in modo che l'asse del motore segua l'alberino del potenziometro.

```
1 #include <Servo.h>
2 Servo servomotorel; // nomino il mio servo "servomotorel"
3 int gradi=0; // memorizzo i gradi dell'asse del servomotore
4
5 void setup() {
6   servomotorel.attach(11); // scrivo dove ho collegato il servomotorel
7
8 }
9
10 void loop() {
11   gradi=analogRead(0)/3.8; // leggo il potenziometro e divido per 3.8 per sapere i gradi
12   // di giro del potenziometro
13   gradi = gradi - 45; // tolgo 45 gradi
14   if (gradi < 0) gradi=0; // se i gradi sono minori di zero metto a zero i gradi
15   if (gradi > 180) gradi = 180; // se i gradi sono maggiori di 180 metto a 180 i gradi
16   servomotorel.write(gradi); // scrivo i gradi nel servomotorel
17   delay(15); // aspetto un tempo per permettere al servomotore di posizionarsi
18 }
19 }
```

PROGRAMACION - IL SERVO MOTORE:

Come prima cosa carichiamo la libreria "Servo" per includere la classe "Servo" nel nostro programma.

In linea 2 dobbiamo dichiarare un 'oggetto' che è il nome che vogliamo dare al nostro servomotore per poterlo richiamare e lo chiamiamo "servomotore1" in modo che se ne volessimo aggiungerne altri li chiameremo "servomotore2" e "servomotore3" in modo che sia semplice sapere quale stiamo azionando.

In linea 3 creiamo una variabile di tipo "int" nella quale memorizziamo la lettura del potenziometro che chiamiamo "gradi". In questa variabile troveremo i gradi di posizionamento del servomotore.

In linea 6, nel "setup()" scriviamo dove è collegato il servomotore1

In linea 11, nel "loop()", scriviamo il valore che leggiamo sull'uscita del potenziometro e lo memorizziamo in "gradi" dividendola per 3.8. La dividiamo per 3.8 perché il potenziometro ha un angolo di rotazione di 270 gradi e il servomotore di 180 e siccome vogliamo che il servomotore segua l'alberino del potenziometro ci serve conoscere i gradi di rotazione.

$$1024 / 270 = 3.8$$

Se sottraiamo i gradi di rotazione del servomotore a quelli del potenziometro otteniamo:

$$270 - 180 = 90$$

e notiamo che il potenziometro ha 90 gradi di rotazione di troppo, 45 gradi a sinistra e 45 a destra che non ci servono.

In linea 13 tagliamo i primi 45 gradi di lettura del potenziometro. Ora se l'alberino del potenziometro si trova a meno di 45 gradi avremo un valore negativo quindi:

In linea 14 mettiamo un controllo di flusso "if" che se la variabile è minore di zero la riportiamo a zero.

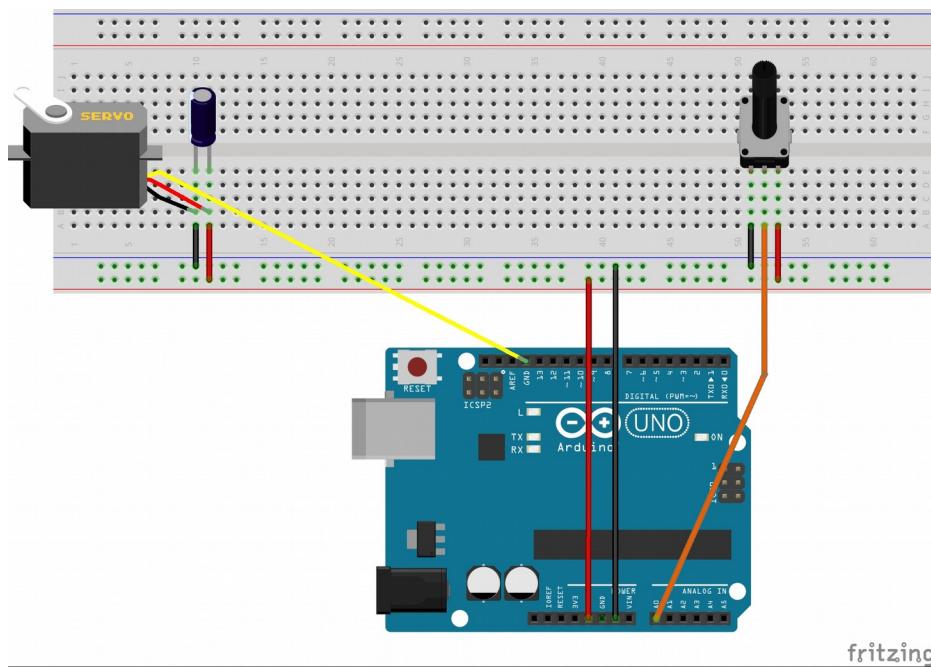
PROGRAMACION - IL SERVO MOTORE:

In linea 15 se l'alberino del potenziometro va oltre a 180 gradi con un altro controllo di flusso "if" la riportiamo a 180 che è il massimo punto che può raggiungere la meccanica del servomotore.

In linea 16 scriviamo il valore di "gradi" nell'oggetto che avevamo chiamato "servomotore1" che è il nostro servomotore.

In linea 17 mettiamo un piccolo delay per permettere al servomotore di raggiungere la posizione.

Possiamo collegare il tutto come in figura.



Per la prima volta vedete nello schema pratico di montaggio un condensatore eletrolitico. Per ora non montatelo e provate il servomotore caricando lo Sketch in Arduino.

Posizioniamo il potenziometro tutto a sinistra e quando il servomotore raggiunge la posizione montate il supporto in dotazione a zero gradi. Ora girando il potenziometro il servomotore lo seguirà.

I CONDENSATORI:

La capacità di carica dei condensatori si misura in Farad che si rappresenta con la lettera "F" in onore a Michael Faraday. Un Farad equivale a 1 Coulomb "C" quando a un condensatore si fornisce una tensione di 1 Volt.

I condensatori si misurano in Farad (F) e nei sottomultipli:

micro Farad (μF) = $1\text{F} / 1,000,000$

nano Farad (nF) = $1\text{F} / 1,000,000,000$

pico Farad (pF) = $1\text{F} / 1,000,000,000,000$

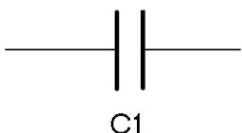
Dire che un condensatore ha una capacità di 1000 pico Farad (pF) è come dire che ha una capacità di 1 nano Farad (nF) e dire che un condensatore ha una capacità di 100 nano Farad (nF) equivale a dire che ha una capacità di 0.1 micro Farad (μF).

Tutte le misure sono multipli di 1000.

Quindi:

$$\begin{aligned}\text{pF} * 1,000 &= \text{nF} \\ \text{nF} * 1,000 &= \mu\text{F}\end{aligned}$$

$$\begin{aligned}\mu\text{F} / 1,000 &= \text{nF} \\ \text{nF} / 1,000 &= \text{pF}\end{aligned}$$

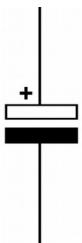


Simbolo elettrico del condensatore non polarizzato

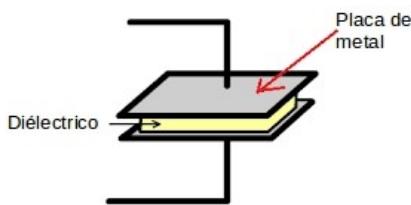


Simbolo elettrico del condensatore variabile

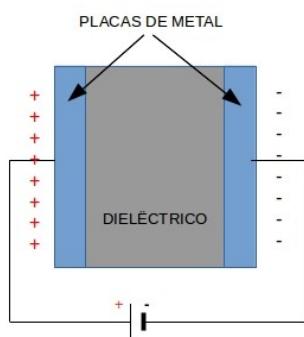
I CONDENSATORI:



Simbolo elettrico del condensatore elettrolitico

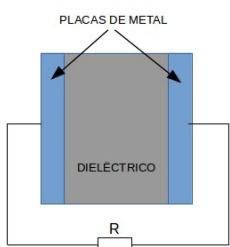


Un condensatore è composto da due placche di metallo separate da un materiale isolante che si chiama dielettrico e che può essere di carta, plastica, ceramica, ossido di tantalio o aria.



Quando forniamo corrente continua a un condensatore gli elettroni si concentrano su una placca di metallo e si fermano perché non possono superare la barriera isolante.

Se scolleghiamo il condensatore gli elettroni restano sulla placca dove si erano accumulati rendendo di fatto il condensatore una piccola batteria. Per scaricarlo e riportare gli elettroni in pari sulle placche dobbiamo collegare in parallelo una resistenza.



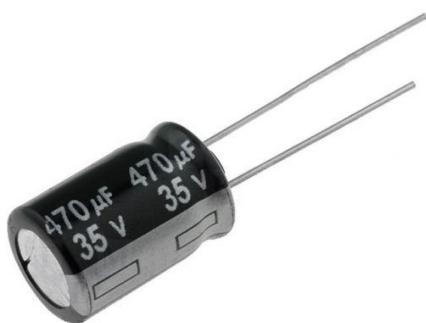
Mettendo una resistenza in parallelo al condensatore chiudiamo il circuito, in questo modo gli elettroni possono migrare verso la placca positiva e il condensatore si scarica come una batteria.

I CONDENSATORI:

Con Arduino si usano molto i condensatori elettrolitici che hanno alte capacità e che devono il loro nome ad un olio elettrolita che usano come dielettrico.

Servomotori e trasmettitori radio richiedono molta corrente di spunto che il condensatore può fornire per il tempo sufficiente senza che si sovraccarichi lo stabilizzatore di tensione di Arduino e senza che questo causi cadute di tensione che bloccherebbero i componenti elettronici.

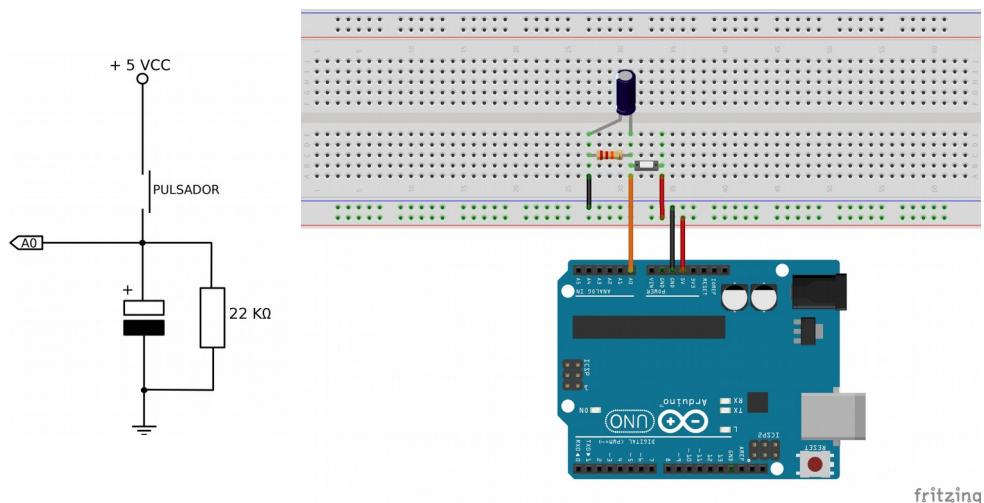
Generalmente si usano condensatori da 470 o 1000 micro Farad (μF).



Il condensatore elettrolitico è polarizzato e la polarità la si riconosce perché è stampata sul suo corpo e dai terminali che come nel LED il positivo è più lungo. I condensatori elettrolitici hanno tensioni di lavoro differenti, è molto importante alimentarli un 30% in meno del loro voltaggio massimo per non vederli esplodere.

Sul loro corpo è scritta la capacità in micro Farad e la tensione massima di lavoro.

con Arduino possiamo vedere la curva di scarica di un condensatore per capire meglio come funzionano.

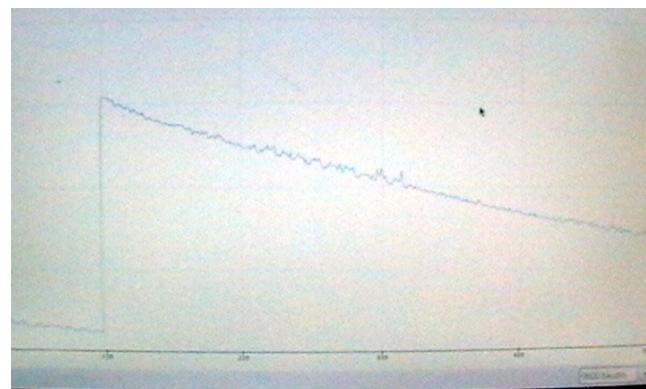


I CONDENSATORI:

Premendo il pulsante il condensatore si carica e contrariamente ad una batteria si carica in un attimo. Quando il pulsante si apre la resistenza in parallelo lo scarica.

Possiamo vedere la curva di scarica nel plotter seriale scrivendo questo semplice programma.

```
1 void setup() {
2   Serial.begin(9600);
3
4 }
5
6 void loop() {
7   Serial.println(analogRead(0));
8   delay(30);
9
10 }
```



Potete provare a cambiare condensatore o la resistenza per vedere quello che succede.

I condensatori hanno valori standard come le resistenze.

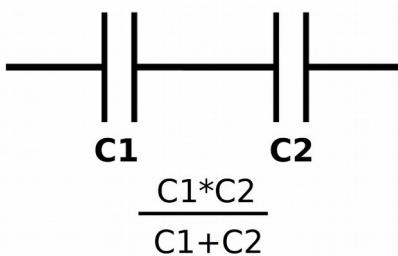
| | | | | | | |
|--------|-------|--------|----------|-----------|------------|------------|
| 1,0 pF | 10 pF | 100 pF | 1.000 pF | 10.000 pF | 100.000 pF | 1,0 microF |
| 1,2 pF | 12 pF | 120 pF | 1.200 pF | 12.000 pF | 120.000 pF | 1,2 microF |
| 1,5 pF | 15 pF | 150 pF | 1.500 pF | 15.000 pF | 150.000 pF | 1,5 microF |
| 1,8 pF | 18 pF | 180 pF | 1.800 pF | 18.000 pF | 180.000 pF | 1,8 microF |
| 2,2 pF | 22 pF | 220 pF | 2.200 pF | 22.000 pF | 220.000 pF | 2,2 microF |
| 2,7 pF | 27 pF | 270 pF | 2.700 pF | 27.000 pF | 270.000 pF | 2,7 microF |
| 3,3 pF | 33 pF | 330 pF | 3.300 pF | 33.000 pF | 330.000 pF | 3,3 microF |
| 3,9 pF | 39 pF | 390 pF | 3.900 pF | 39.000 pF | 390.000 pF | 3,9 microF |
| 4,7 pF | 47 pF | 470 pF | 4.700 pF | 47.000 pF | 470.000 pF | 4,7 microF |
| 5,6 pF | 56 pF | 560 pF | 5.600 pF | 56.000 pF | 560.000 pF | 5,6 microF |
| 6,8 pF | 68 pF | 680 pF | 6.800 pF | 68.000 pF | 680.000 pF | 6,8 microF |
| 8,2 pF | 82 pF | 820 pF | 8.200 pF | 82.000 pF | 820.000 pF | 8,2 microF |

Come potete vedere dalla tabella si tratta di soli 12 valori che si ripetono moltiplicandosi per 10.

I CONDENSATORI:

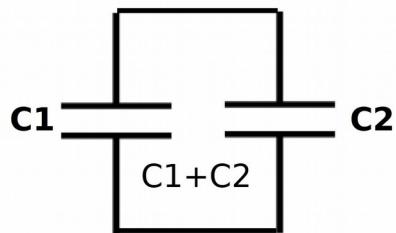
Come le resistenze anche i condensatori si possono collegare in serie e in parallelo.

IN SERIE



Mettendo in serie due condensatori otteniamo un valore di capacità minore del condensatore più piccolo.

IN PARALLELO



Mettendo in parallelo due condensatori la capacità si somma ma la tensione di lavoro si dimezza.

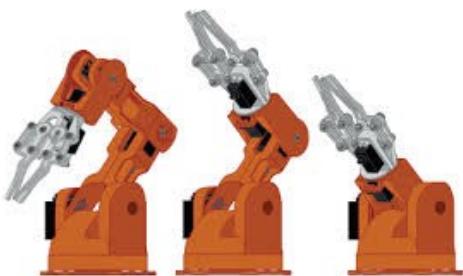
I condensatori sono molto usati in elettronica analogica come filtri, nei circuiti oscillatori nella gestione dei segnali audio e molto altro.

Il condensatore da 1000 micro Farad in parallelo all'alimentazione del servomotore serve per fornire la corrente di spunto per il motore elettrico al suo interno. E siccome un servomotore si muove continuamente in avanti e indietro seguendo il codice del programma tutti questi spunti anche se tollerati dal circuito di stabilizzazione di Arduino lo porterebbero a scaldarsi molto rapidamente.

Ora potete capire per quale motivo ho messo un condensatore in parallelo all'alimentazione del servomotore.

IL SERVO MOTORE:

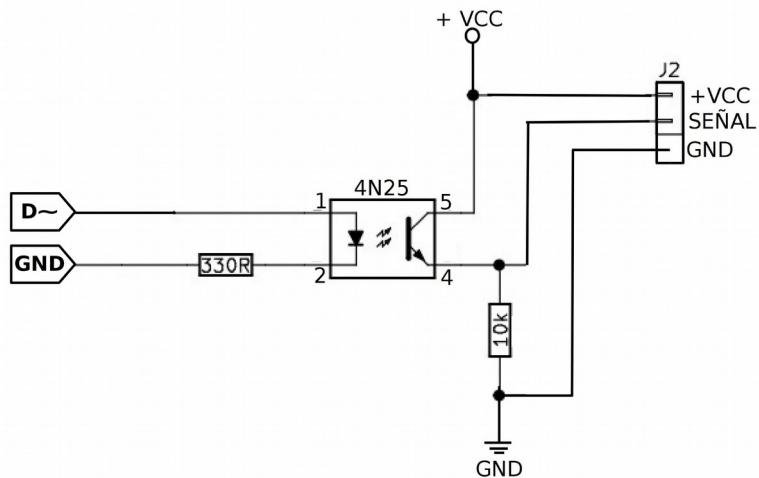
Con il servomotore si possono fare molte cose e molte volte ne usiamo molti insieme per un progetto di robotica.



In questo caso non possiamo usare il 5 Volt di arduino perché non avrebbe la corrente necessaria per muovere più servomotori quindi dobbiamo usare una alimentazione secondaria adatta al tipo di servomotore che abbiamo intenzione di usare.

Come scrissi i motori in corrente continua disturbano molto il micro controllore quindi è meglio usare un foto accoppiatore per il segnale di controllo in modo che non ci siano collegamenti tra la tensione del micro controllore e quella dei motori.

Possiamo usare lo schema che abbiamo usato per comandare il motore in corrente continua.



In questo modo possiamo collegare tutti i servomotori che vogliamo ripetendo questo schema per ogni servomotore.

PROGRAMMA COMPLETO DEL GIOCO:

```
1 #define pulsante_1 2
2 #define LED_1 3
3 #define LED_BLU 4
4 #define LED_2 5
5 #define pulsante_2 6
6 byte inizio=0; // quando inizia il gioco la variabile va a 1
7 int valore=0; // in "valore" metto un numero casuale del tempo di attesa
8 unsigned long attesa=0; // memorizzo il valore del contatore dei millisecondi per calcolare l'attesa
9 byte rifare=0; // se un giocatore bari la variabile va a 1
10 byte punteggio_1=0; // memorizzo il punteggio del giocatore 1
11 byte punteggio_2=0; // memorizzo il punteggio del giocatore 2
12 byte gara_finita=0; // va a 1 quando finisce una gara
13 byte partite=0; // memorizzo il numero delle partite
14
15 void setup(){
16   pinMode(pulsante_1, INPUT);
17   pinMode(LED_1, OUTPUT);
18   pinMode(LED_BLU, OUTPUT);
19   pinMode(LED_2, OUTPUT);
20   pinMode(pulsante_2, INPUT);
21 }
22 }
23
24 void loop(){
25   if(inizio==0){ // se è a zero è una nuova sfida
26     inizio_gioco(); // chiamo la funzione per il gioco di luci
27   }
28   partite=partite+1; // sommo 1 per tenere il conto delle partite
29   tempo(); // chiamo la funzione per il numero casuale dell'attesa
30   attesa_tempo(); // chiamo la variabile che controlla che nessuno bari nell'attesa
31   if(rifare==0){ // se "rifare" è - a 0 è perché nessuno ha barato
32     digitalWrite(LED_BLU,HIGH); // accendo il led blu per iniziare la sfida
33     gara();
34     if(partite>10){ // se il numero delle partite è 11 il gioco finisce
35       fine(); // vado a vedere chi ha vinto
36       punteggio_1=0; // metto a zero il punteggio del giocatore 1 per una nuova gara
37       punteggio_2=0; // metto a zero il punteggio del giocatore 2 per una nuova gara
38       inizio=0; // metto a zero la variabile per iniziare un nuovo gioco
39     }
40   }
41 }
42 rifare=0; // metto a zero la variabile per iniziare una nuova partita
43 }
44
45 void inizio_gioco(){ // questa funzione fa il gioco di luci dall'inizio
46
47   inizio=1; // metto a 1 la variabile per non rifare il gioco di luci fino alla fine
48   for(int i=0; i<10; i++){ // ripeto 10 volte il gioco di luci
49     digitalWrite(LED_1, HIGH);
50     delay(150);
51     digitalWrite(LED_1, LOW);
52     digitalWrite(LED_BLU, HIGH);
53     delay(150);
54     digitalWrite(LED_BLU, LOW);
55     digitalWrite(LED_2, HIGH);
56     delay(150);
57     digitalWrite(LED_2, LOW);
58     digitalWrite(LED_BLU, HIGH);
59   }
60 }
```

PROGRAMMA COMPLETO DEL GIOCO:

```
59     delay (150);
60     digitalWrite ( LED_BLU, LOW );
61 }
62 delay ( 1000 );
63 digitalWrite ( LED_1, HIGH );
64 digitalWrite ( LED_2, HIGH );
65 delay ( 3000 );
66 digitalWrite ( LED_1, LOW );
67 digitalWrite ( LED_2, LOW );
68
69 }
70
71 void tempo(){ // questa funzione trova il tempo di attesa
72
73     randomSeed(analogRead(0)); // creo il numero generatore dei numeri casuali
74     valore=random(500,5000); //memorizzo il numero casuale da 500 a 5000
75 }
76
77 void attesa_tempo() { // questa funzione controlla se un giocattore bара nel'attesa
78
79     attesa = millis(); // memorizzo il numero del contatore dei millisecondi
80     attesa = attesa + valore; // sommo al tempo letto del contatore il mio tempo di attesa
81
82     while ( attesa > millis() ){ // se la somma è minore del valore del contatore
83         if ( digitalRead ( pulsante_1 ) == HIGH ) { // guardo se il pulsante 1 è premuto
84             punti_2++; // se è premuto metto un punto al giocatore 2
85             rifare = 1; // porto a 1 la variabile di controllo per rifare la partita
86         }
87         if ( digitalRead ( pulsante_2 ) == HIGH ) { // guardo se il pulsante 2 è premuto
88             punti_1++; // se è premuto metto un punto al giocatore 1
89             rifare = 1; // porto a 1 la variabile di controllo per rifare la partita
90         }
91     }
92 }
93
94 void punti_1(){ // questa funzione aggiunge un punto al giocattore 1
95
96     punteggio_1 = punteggio_1 +1; // sommo 1 ai punti del giocattore 1
97
98     for ( int i=0; i<4; i++){ // faccio lampeggiare 4 volte il led del giocattore 1
99         digitalWrite ( LED_1, HIGH );
100         delay(200);
101         digitalWrite ( LED_1, LOW );
102         delay ( 200 );
103     }
104     digitalWrite ( LED_BLU, LOW ); // spengo il led blu
105 }
106
107 void punti_2(){ // questa funzione aggiunge un punto al giocattore 2
108
109     punteggio_2 = punteggio_2 +1; // sommo 1 ai punti del giocattore 2
110
111     for ( int i=0; i<4; i++){ // faccio lampeggiare 4 volte il led del giocattore 2
112         digitalWrite ( LED_2, HIGH );
113         delay(200);
114         digitalWrite ( LED_2, LOW );
115         delay ( 200 );
116     }
```

PROGRAMMA COMPLETO DEL GIOCO:

```
117  digitalWrite (LED_BLU, LOW); // spengo il led blu
118 }
119
120 void gara(); // questa funzione controlla chi premre per primo il pulsante
121
122 while ( gara_finita == 0 ){ // il "while" si ferma quando cambia di valore la variabile
123   if ! digitalRead ( pulsante_1 ) == HIGH ){ // se il giocatore 1 preme per primo
124     gara_finita = 1; // metto a 1 la variabile di controllo per fermare il "while"
125     punti_1()); // aggiungo un punto al giocatore 1
126   }
127
128   if !gara_finita == 0 ){ // se il giocatore 1 non ha premuto controllo al secondo
129
130     if ! digitalRead ( pulsante_2 ) == HIGH ){ // se il giocatore 2 presiona per primo
131       gara_finita = 1; // metto a 1 la variabile di controllo per fermare il "while"
132       punti_2()); // aggiungo un punto al giocatore 2
133     }
134   }
135 }
136 gara_finita=0;
137 }
138 void fine(){
139   if !punteggio_1>punteggio_2){ // se il punteggio del giocatore 1 è maggiore
140     for:int i=0: i<4: i++){ // faccio lampeggiare il led blu e il led rosso
141       digitalWrite(LED_BLU,HIGH); // del giocatore 1 per 4 volte
142       digitalWrite(LED_1,LOW);
143       delay(1000);
144       digitalWrite(LED_BLU,LOW);
145       digitalWrite(LED_1,LOW);
146       delay(500);
147     }
148   }
149
150 else { // se non è maggiore vince il giocatore 2
151   for:int i=0: i<4: i++){ // faccio lampeggiare il led blu e il led rosso
152     digitalWrite(LED_BLU,HIGH); // del giocatore 2 per 4 volte
153     digitalWrite(LED_2,HIGH);
154     delay(1000);
155     digitalWrite(LED_BLU,LOW);
156     digitalWrite(LED_2,LOW);
157     delay(500);
158   }
159 }
160 partite = 0; // metto a zero il numero delle partite giocate
161 delay ( 5000 ); // aspetto 5 secondi prima di ricominciare un nuovo gioco
162 }
163 }
```


PROGRAMMA COMPLETO CICLO-COMPUTER:

```
1 #include <OneWire.h>
2 #include <DallasTemperature.h>
3 #define DQ 9 // è la connessione DQ del DS18B20
4 OneWire oneWire(DQ); // comunica a la libreria "OneWire" la connessione del DS18B20
5 DallasTemperature sensorDS18B20(oneWire); // comunica a la libreria "DallasTemperature" la connessione del DS18B20
6
7
8 #include <Wire.h>
9 #include <LiquidCrystal_I2C.h>
10 LiquidCrystal_I2C lcd(0x3F,16,2); // metto i dati del display
11
12 byte velocita=0; // memorizzo la velocità in kmh
13 float gradi=0; // memorizzo la lettura della temperatura
14 byte ore=0; // memorizzo le ore di viaggio
15 byte minuti=0; // memorizzo i minuti di viaggio
16 byte secondi=0; // memorizzo i secondi di viaggio
17 long tempo=0; // memorizza in secondi il tempo di viaggio
18 float tmp=0; // è una variabile temporale per il calcolo delle ore, minuti, secondi
19
20 #define sensore 3 // dove collego il sensorereed
21 unsigned long x_nuovo=0; // memorizza il valore di nullissimo quando poso il magnete davanti al sensore
22 unsigned long x_vecchio=0; // memorizza il tempo dello lettore precedente
23 int tempo_deli_giro=0; // memorizza il tempo di un giro della ruota
24 const circuitterza=1.4; // diametro in metri della ruota
25 byte v=0; // è la variabile di controllo se si ferma le ruote
26 unsigned long zl=0; // memorizza il valore di nullissimo per vedere se le ruote è fermo
27
28 #define DISPLAY 5 // è il collegamento del led del display
29 #define toto_resistenza 0 // è il collegamento delle fotoresistenza
30
31 void setup()
32 {
33   lcd.init(); // inizializzo il display
34   lcd.backlight(); // accendo il led del display
35   sensorDS18B20.begin(); //inizializzo la libreria del sensore di temperatura
36   pinMode(sensore,INPUT_PULLUP); // imposto come ingresso il pin del sensore read
37   attachInterrupt(0, Km(), RISING); // attivo l'interrupt che chiama la funzione Km() quando l'ingresso 0 va basso
38
39 }
40
41
42 void loop()
43 {
44   lcd.setCursor(0, 0); // metto il cursore in alto a sinistra
45   lcd.print("Kmh "); // scrivo Km più uno spazio
46
47   if (velocita<10) lcd.print("0"); // se la velocità è minore di 10 scrivo uno 0
48   lcd.print(velocita); // scrivo il valore della velocità
49   lcd.setCursor(5,0); // metto il cursore in posizione 10 nella prima linea
50   lcd.print("° "); // scrivo "°" più uno spazio
51   if (gradi<10) lcd.print("0"); // se gradi è minore di 10 scrivo uno 0
52   lcd.print(gradi); // scrivo i gradi
53   lcd.setCursor(5,1); // metto il cursore in posizione 3 nella
54   lcd.print("h "); // scrivo "h" più uno spazio
55   if (ore<10) lcd.print("0"); // se il valore delle ore è minore di 10 scrivo uno 0
56   lcd.print(ore); // scrivo il valore delle ore
57   lcd.print(":"); // scrivo ":"
58   if (minuti<10) lcd.print("0"); // se il valore dei minuti è minore di 10 scrivo uno 0
```

PROGRAMMA COMPLETO CICLO-COMPUTER:

```
59 lcd.print(minuti);           // scrivo il valore dei minuti
60 lcd.print(":");             //scrivo ":" 
61 if (secondi<10) lcd.print("0");// se il valore dei secondi è minore di 10 scrivo uno 0
62 lcd.print(secondi);         // scrivo il valore dei secondi
63 tempo_di_viaggio();        // chiamo la funzione "tempo di viaggio"
64 temperatura();             // chiamo la funzione per leggere la temperatura
65 if (c==1) c=millis();        // controllo se la ruota è ferma memorizzando il valore di millis()
66 if (c==0){                  // se la variabile di controllo è 0 è perché la funzione Kmh non si è eseguita
67   if((sl(2000)<millis()) velocita=0; // se dopo due secondi non si esegue la funzione Kmh è perché
68 }                           // la ruota è ferma e metto a 0 la velocità
69 c=t;                        // metto a 1 la variabile di controllo
70
71 if (! analogRead(foto_resistenza)/4) analogWrite(DIPLAY,60); // mette un limite alla luce del display
72 if (! analogRead(foto_resistenza)/4) analogWrite(DIPLAY, (analogRead(foto_resistenza)/4));
73
74
75 void tempo_di_viaggio(){    // calcolo il tempo di viaggio
76   tempo = millis() / 1000; // memorizzo il tempo in secondi
77   tmp = tempo / 3600; // divide i secondi per 3,600 per trovare le ore
78   ore = tmp; // memorizzo la parte intera della divisione
79   tempo = tempo - (ore * 3600); // moltiplico la parte intera per 3,600 e la sottraggo al totale dei secondi per trovare
80   tmp = tempo / 60; // dividere i secondi rimanenti per 60 per calcolare i minuti
81   minuti = tmp; //memorizzo la parte intera del calcolo dei minuti
82   secondi = tempo - (minuti * 60); // moltiplico i minuti per 60 e la sottraggo dal totale per trovare i secondi
83
84 }
85
86 void temperatura(){ // leggo la temperatura
87   sensorDS18B20.requestTemperatures(); // richiedo la lettura della temperatura alla libreria
88   gradi = sensorDS18B20.getTempCByIndex(0); // memorizzo in gradi la lettura della libreria
89
90
91 void Kmh(){ // calcolo la velocità
92
93   x_nuovo=millis(); // in x nuovo memorizzo il valore di millis()
94   if (x_nuovo>x_vecchio+6){ // se il valore nuovo è minore del valore vecchio più 6 millisecondi è un rumore del se
95     // e non è valida la lettura, se è maggiore la lettura è valida
96   tempo_deli_giro=x_nuovo-x_vecchio; // il tempo del giro della ruota è la differenza del tempo nuovo - il vecchio
97   x_vecchio=x_nuovo; // memorizzo il valore nuovo nel vecchio per prendersi un nuovo valore
98 }
99
100 velocita=(cicloinferenza*3600)/tempo_deli_giro;// calcolo la velocità
101 c=1;                         // metto a 1 la variabile di controllo
102
103 }
```

LO STARTER KIT

Per realizzare le esperienze pratiche di questo libro serve avere:

- 1 Arduino UNO
- 1 basetta di prova
- 1 display I2C 16 x 2
- 1 potenziometro lineare da 4.7 KΩ
- 2 LED rossi
- 2 LED verdi
- 2 LED gialli
- 1 LED blu
- 10 resistenze de 220 Ω
- 5 resistenze de 330 Ω
- 5 resistenze de 2.2 KΩ
- 5 resistenze de 4.7 KΩ
- 10 resistenze de 10 KΩ
- 5 resistenze de 100 KΩ
- 1 foto- resistenza
- 2 pulsanti
- 1 sensore Dallas 18B20
- 1 sensore magnetico da porta filare
- 1 condensatore elettrolitico da 15V 470 μF
- 1 condensatore elettrolitico da 15V 1000 μF
- 2 transistor BC337
- 2 MOSFET IRF540N
- 1 relé 12V DC
- 1 motore 12V DC
- 2 diodi 1N4004 o 1N4007
- 1 fotoaccoppiatore 4N25 o equivalente
- cavi tipo DUPONT MM di molti colori
- cavi tipo DUPONT MF di molti colori