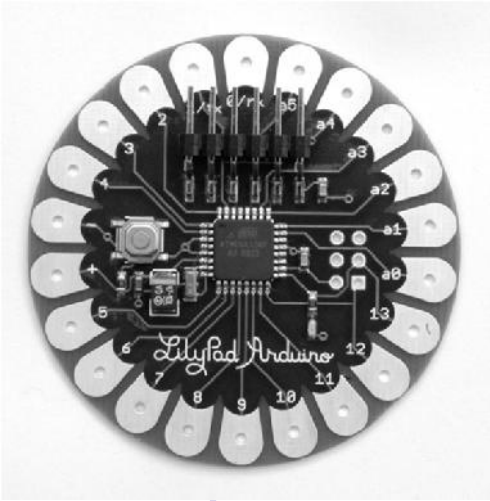


Progetti reali con ARDUINO

Introduzione alla scheda Arduino (parte 2ª)

gennaio 2015 – Giorgio Carpignano
I.I.S. PRIMO LEVI
C.so Unione Sovietica 490 (TO)

Materiale didattico:
www.istitutoprimolevi.gov.it

Il menù per oggi

Physical computing

Schede “add-on” (shield) da “aggiungere” a Arduino

Lettura dei pulsanti

Comunicazione con altri dispositivi

Le istruzioni fondamentali del linguaggio C

- **if …. else**
- **while()**
- **do …. while()**
- **for**
- **switch …. case**

Tipi di variabili e costanti

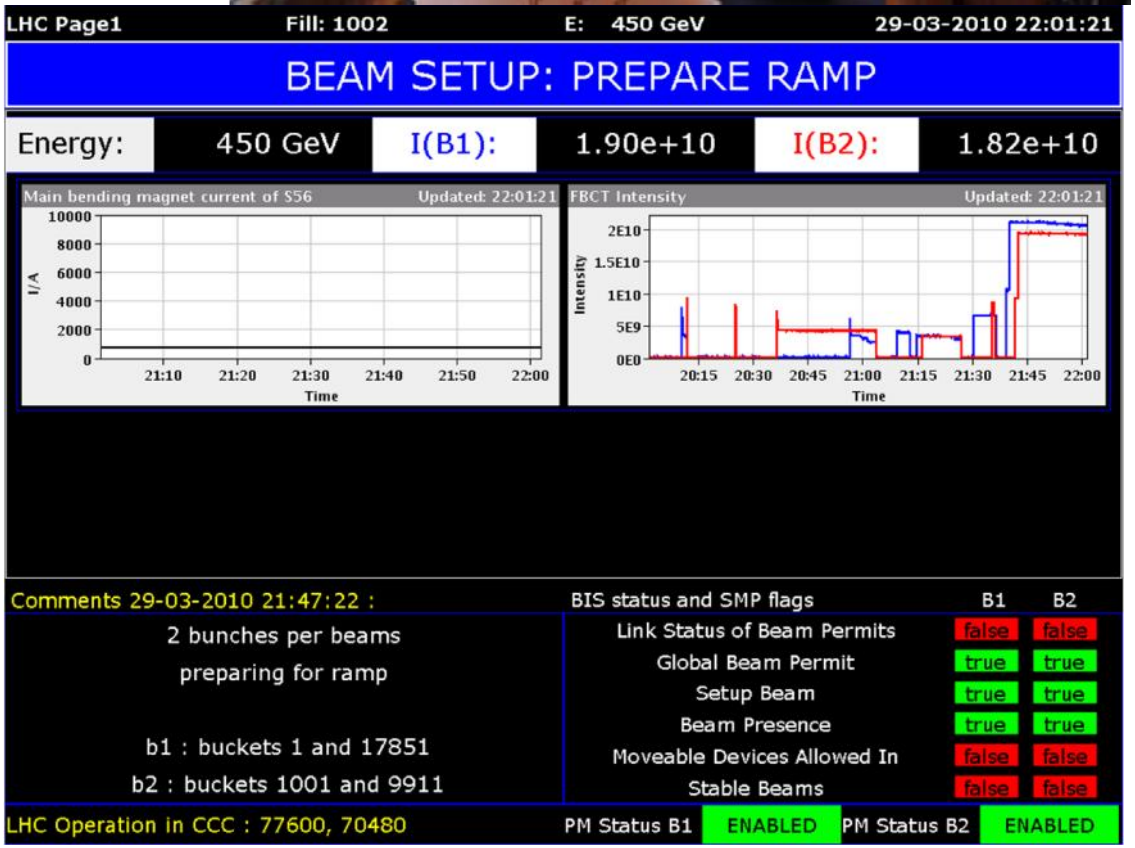
Logica digitale AND, OR, NOT, EX-OR

Inserimento dati da tastiera del Computer

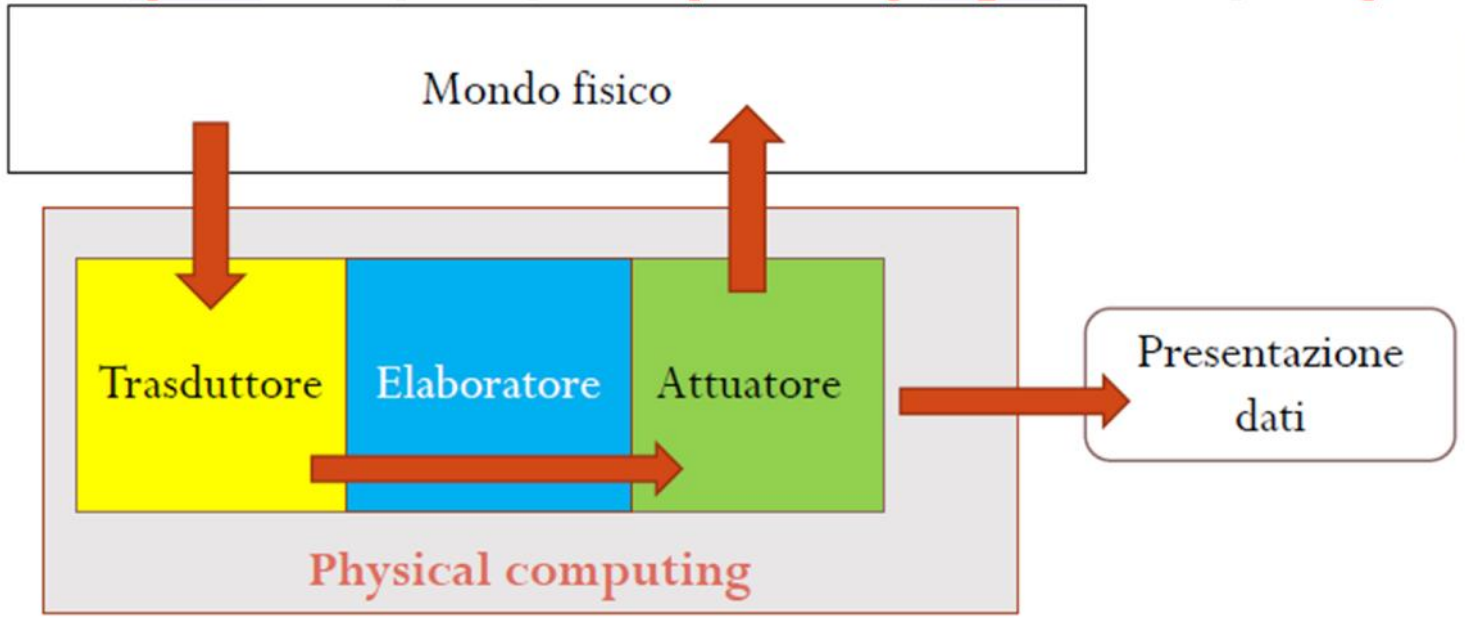
Scheda Arduino in modalità “Stand-alone”

Physical computing

- Physical computing, permette il controllo di un sistema fisico reale utilizzando delle risorse hardware / software e interagendo con una realtà analogica fornita di infiniti valori.



Il dispositivo ideale per il physical computing



Trasduttore/i che non interferisce con i fenomeni e non altera la misura, sensibilità, accuratezza, riproducibilità, frequenza di campionamento, immunità al rumore, selettività, etc

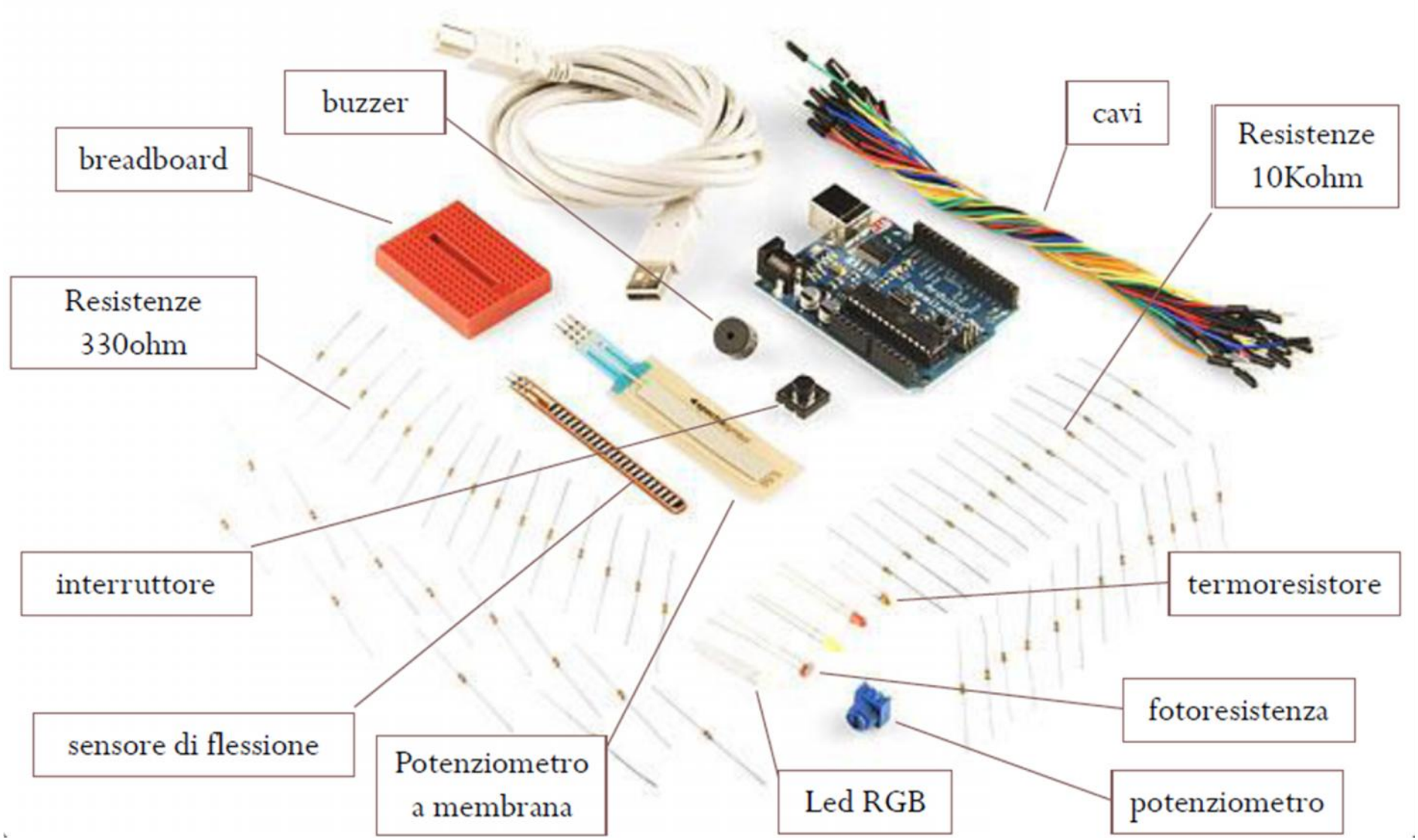
Attuatore/i con caratteristiche ideali analoghe al trasduttore, potenza illimitata in uscita, etc

Elaboratore/i con memoria illimitata, capacità di calcolo illimitata, etc

La realtà è differente, ogni miglioramento aumenta i costi, tuttavia, a patto di accettare compromessi ragionevoli, sono disponibili in commercio strumenti a costi accessibili (quindi interessanti anche per un utilizzo puramente didattico)

Arduino starter kit

http://store.arduino.cc

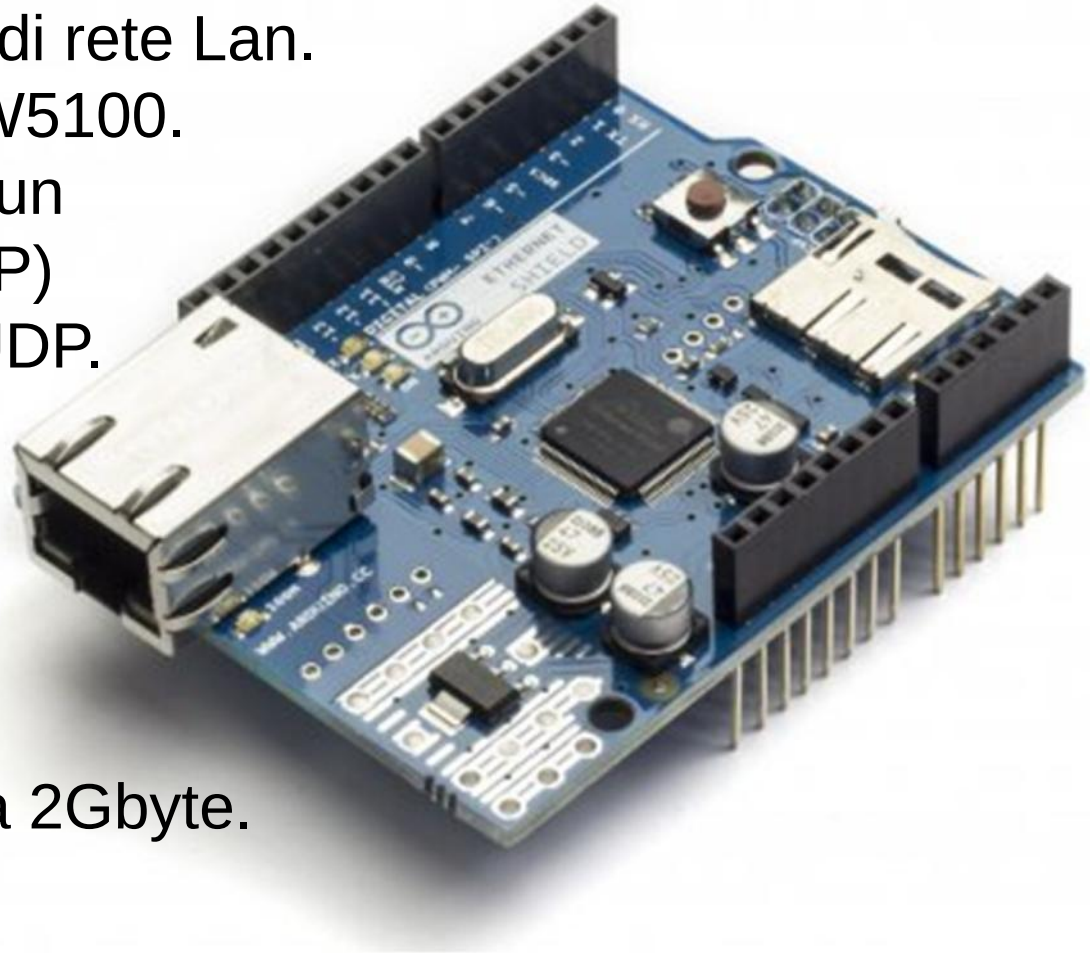


ARDUINO → scheda add-on → SD shield

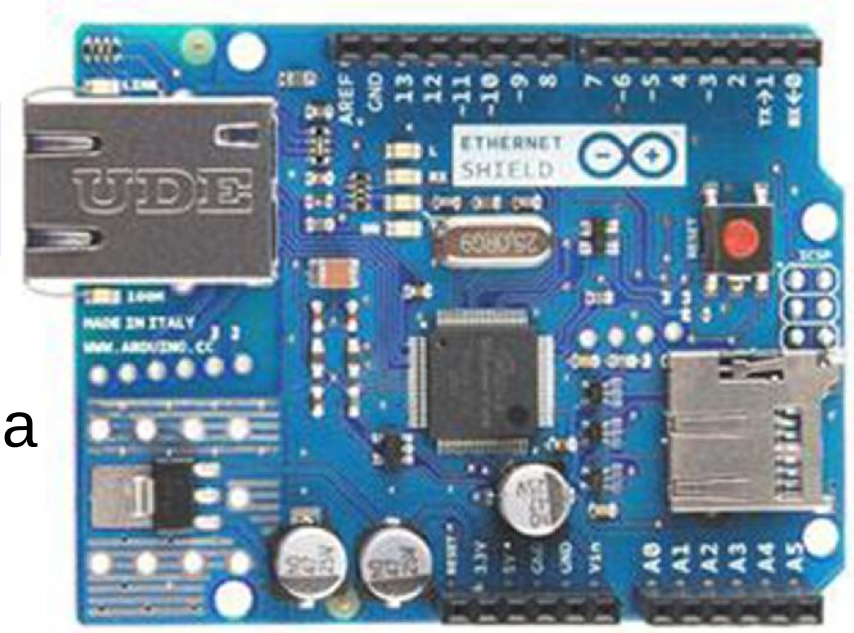
- Permette ai dispositivi come Arduino di leggere e scrivere le **SD card** con apposite librerie disponibili gratuitamente.
- **SD shield** è particolarmente interessante per la realizzazione di semplici **data logger** (acquisizione e memorizzazione di segnali analogici su più canali)

ARDUINO → scheda add-on → Arduino Ethernet Shield Rev3 (29€)

- Permette alla scheda Arduino di connettersi a internet tramite cavo di rete Lan.
- Utilizza il chip Wiznet W5100.
- Provvede ad utilizzare un collegamento di tipo (IP) con protocollo TCP e UDP.
- Supporta fino a 4 connessioni contemporanee.
- Contine una microSD card con possibilità di memorizzare dati fino a 2Gbyte.



ARDUINO → ETHERNET shield (29€)



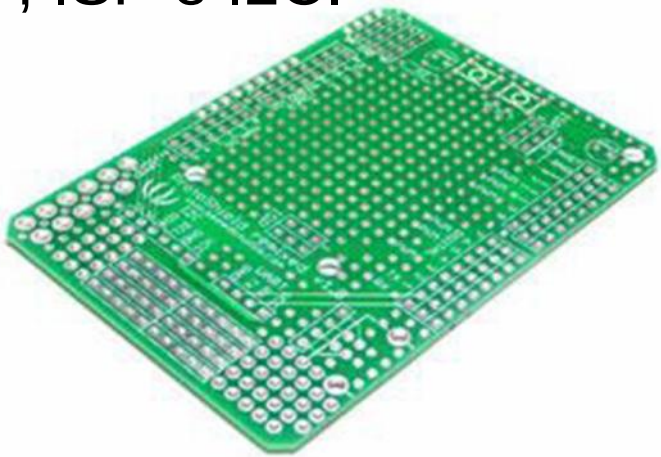
- Dispositivo basato sul chip ethernet WiznetW5100
- Permette di connettere una scheda Arduino ad una LAN utilizzando la libreria Ethernet library
- Dispone di connettore per micro SD card
- Supporta fino a quattro connessioni socket simultanee
- I pin digitali 10, 11, 12 e 13 di Arduino vengono impiegati per comunicare con la scheda, quindi quando si utilizza questa scheda, questi pin non possono essere utilizzati come I/O
- Il pulsante di reset sulla scheda resetta entrambe le schede
- Compatibile anche con la scheda Arduino MEGA
- PowerOver Ethernet utilizzabile (alimentazione tramite rete Lan)

ARDUINO → Wi-Fi Shield (69€)

- Modulo **transceiver** (trasmettitore + ricevitore) **Wi-Fi** a 2,4 GHz standard 802.11 IEEE della Microchip
- Antenna integrata su PCB e supporto integrato per l'hardware AES, TKIP e (**WEP, WPA , WPA2**)
- Copertura di qualche decina di metri
- Consente l'uso di micro SD card
- Alimentazione 12 V fornita direttamente dalla scheda Arduino

ARDUINO → PROTO Shield

- Piastra sperimentale (58,50 x 82,70 mm) per piccole applicazioni, realizzata appositamente per le schede Arduino
- Permette di avere un numero maggiore di piazzole su cui montare i componenti.
- Alcune piazzole sono predisposte per montare un connettore USB tipo B, un mini potenziometro da stampato, pulsanti, LED, ecc.
- Dispone di piazzole riservate al montaggio di connettori per UART, ISP e I2C.

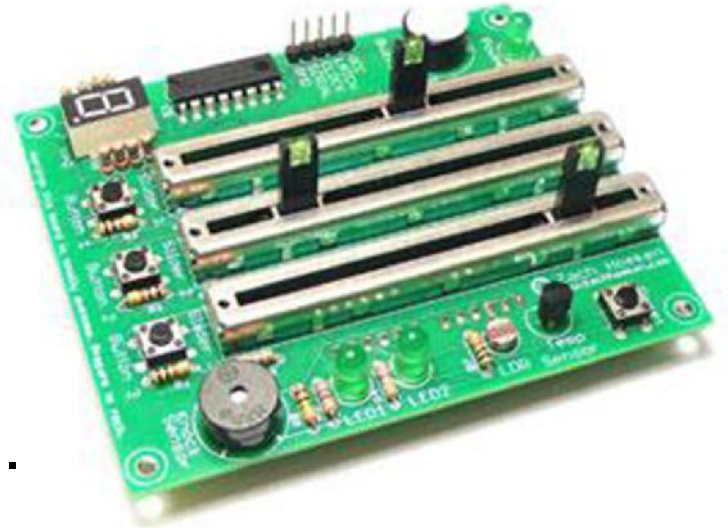


ARDUINO → DANGER Shield

- Montata sopra una scheda Arduino, permette di testare i vari ingressi / uscite
- Viene fornita in kit (va montata)

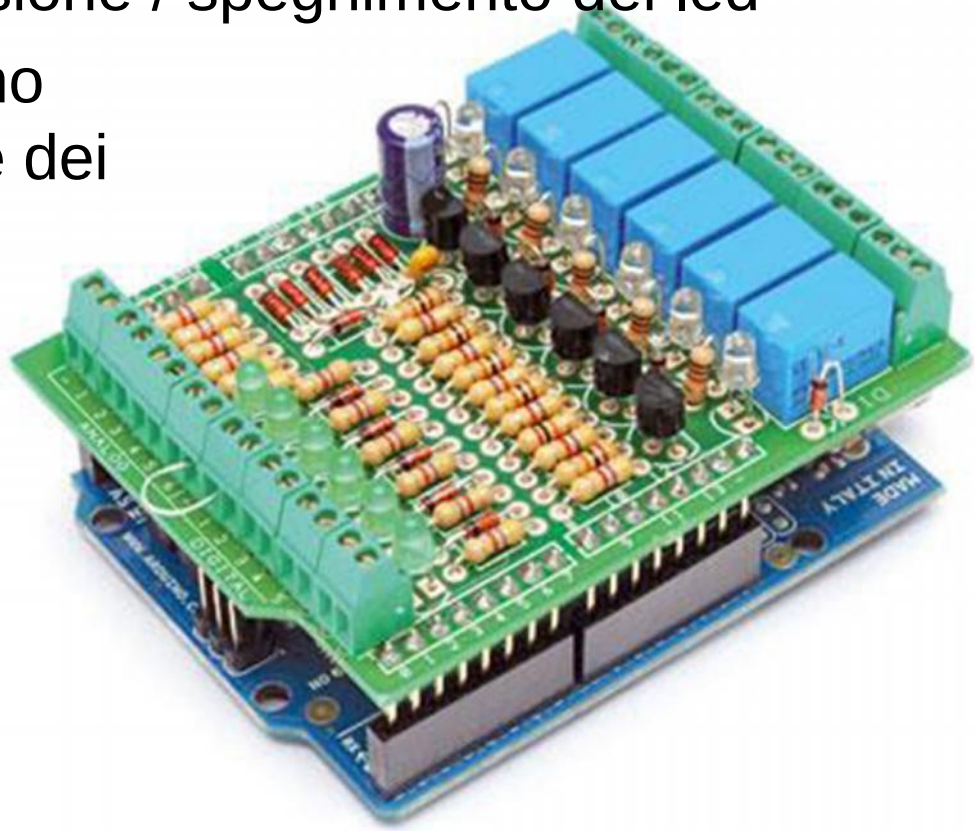
Il KIT contiene:

- 3 slider (potenziometri lineari) con Led integrati, 4 pulsanti
- 3 led indicatori, 1 buzzer
- 1 sensore di temperatura
- 1 fotoresistenza
- 1 knocksensor (sensore di tilt)
- 1 display a 7 segmenti
- 1 integrato 74HC595N (shiftregister).



ARDUINO → IN/OUT Shield

- 6 uscite con relè a libero scambio
- 6 ingressi digitali in tensione TTL compatibile (0 = GND e 1 = 5V)
- 6 ingressi analogici in tensione da 0 a 5V
- Lo stato logico degli ingressi digitali e delle uscite dei relè è visibile tramite accensione / spegnimento dei led
- Le linee di input / output sono collegate ad Arduino tramite dei corrispondenti pin-strip da 2.54 mm
- I relè utilizzati devono essere alimentati a 12V

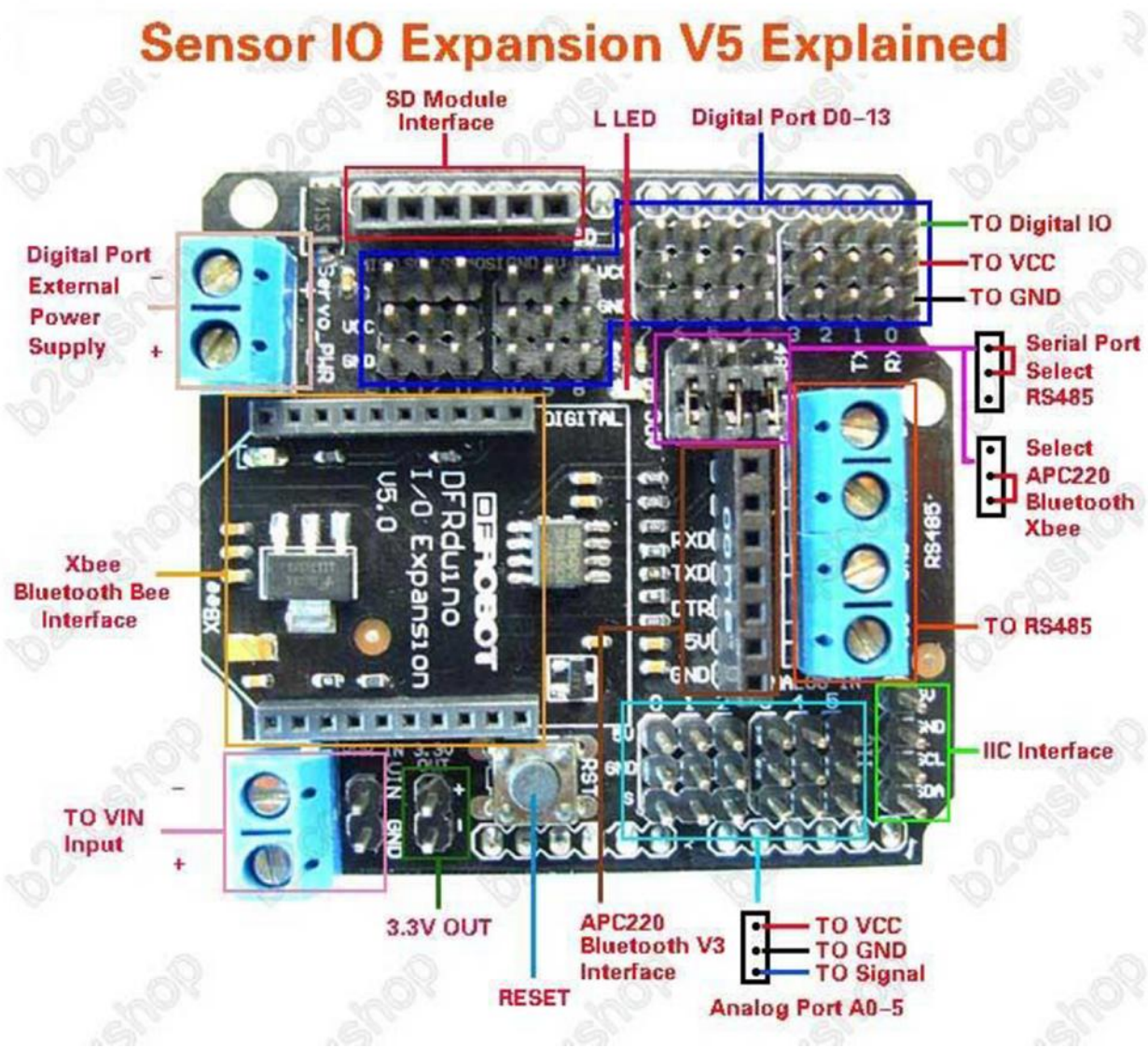


ARDUINO → Graphic LCD4884 Shield

- Display alfanumerico e grafico con 84 x 48 punti.
- Alimentazione: 5V
- Interfaccia SPI (utilizza i pin digitali di Arduino: 2, 3, 4, 5, 6)
- Pulsante di reset presente sulla scheda
- Integra un joystick con 5 gradi di libertà (utilizza il pin analogico di Arduino: 0)
- Integra 6 Digital I/O e 5 Analog I/O per una facile prototipizzazione
- Controllo della luminosità del display (utilizza il pin digitale di Arduino: 7)

ARDUINO → I/O Expansion Shield

- Supporta protocollo di comunicazione RS485
- Supporta Xbee (Xbee pro)
- Supporta Bluetooth
- Supporta APC220
- Supporta SD card read/write



ARDUINO → Motor shield FE

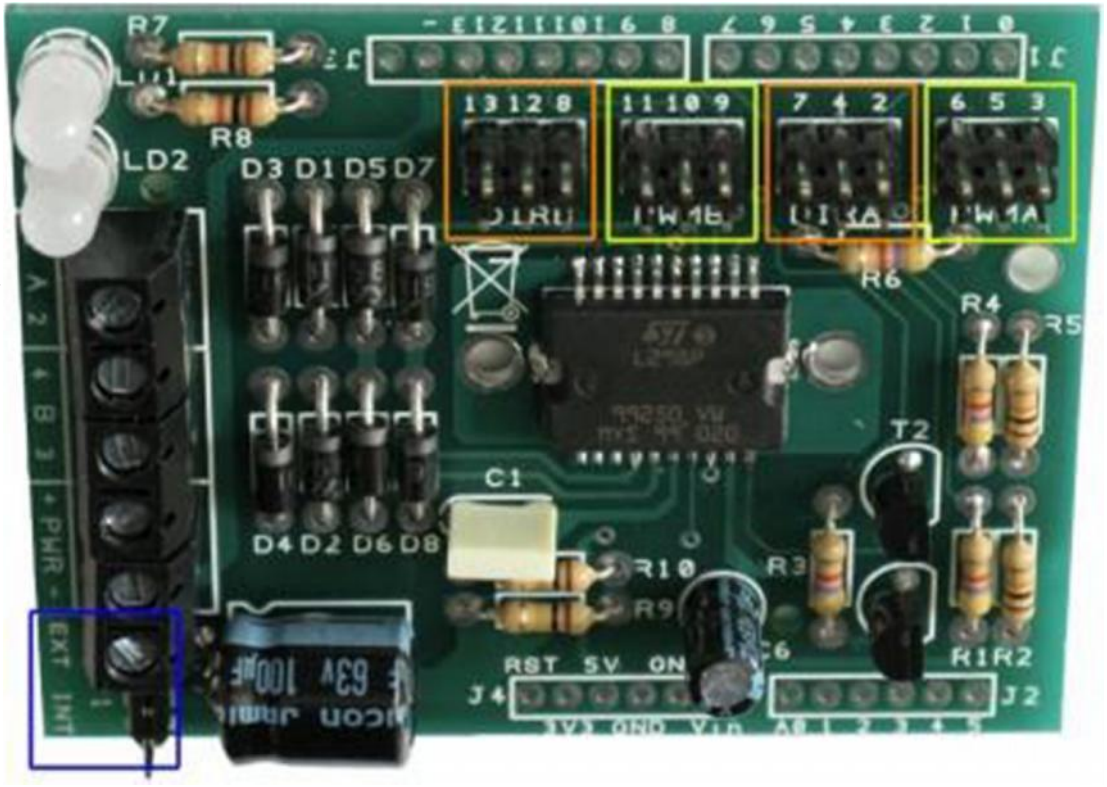
- Shield per controllare 2 motori in corrente continua (in kit)
- Shield basata sul driver doppio full bridge (integrato L298P)
- Ciascuno dei due canali d'uscita dello shield è in grado di fornire una corrente massima 2A
- è possibile definire mediante jumper:

  DIRB direzione motore B

  PWMB potenza motore B

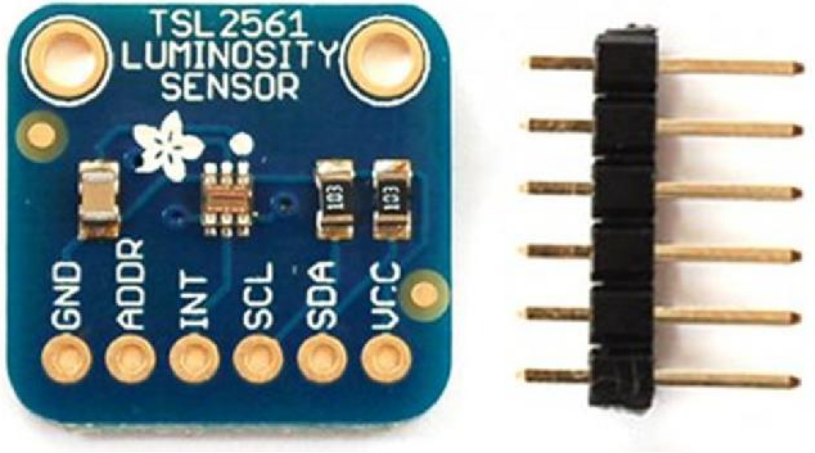
  DIRA direzione motore A

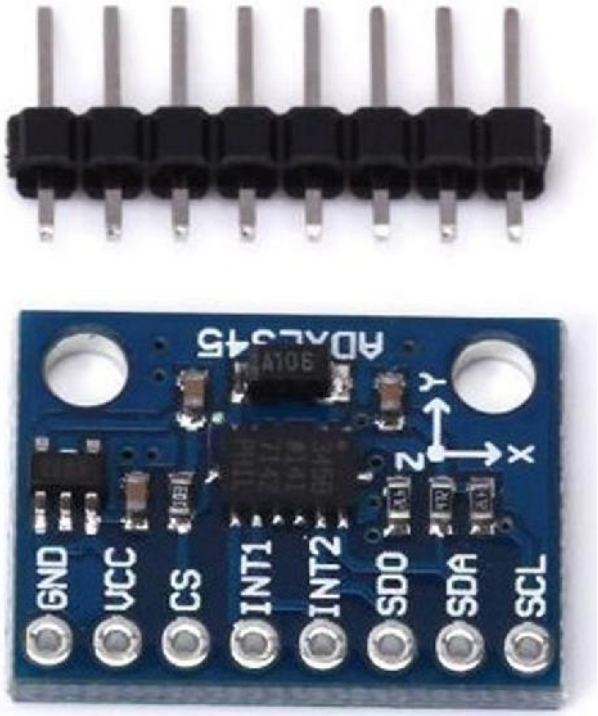
  PWMA potenza motore A



ARDUINO →
Sensori / Attuatori / Componenti

TSL2561 digital luminosity / light sensor (5,60€)





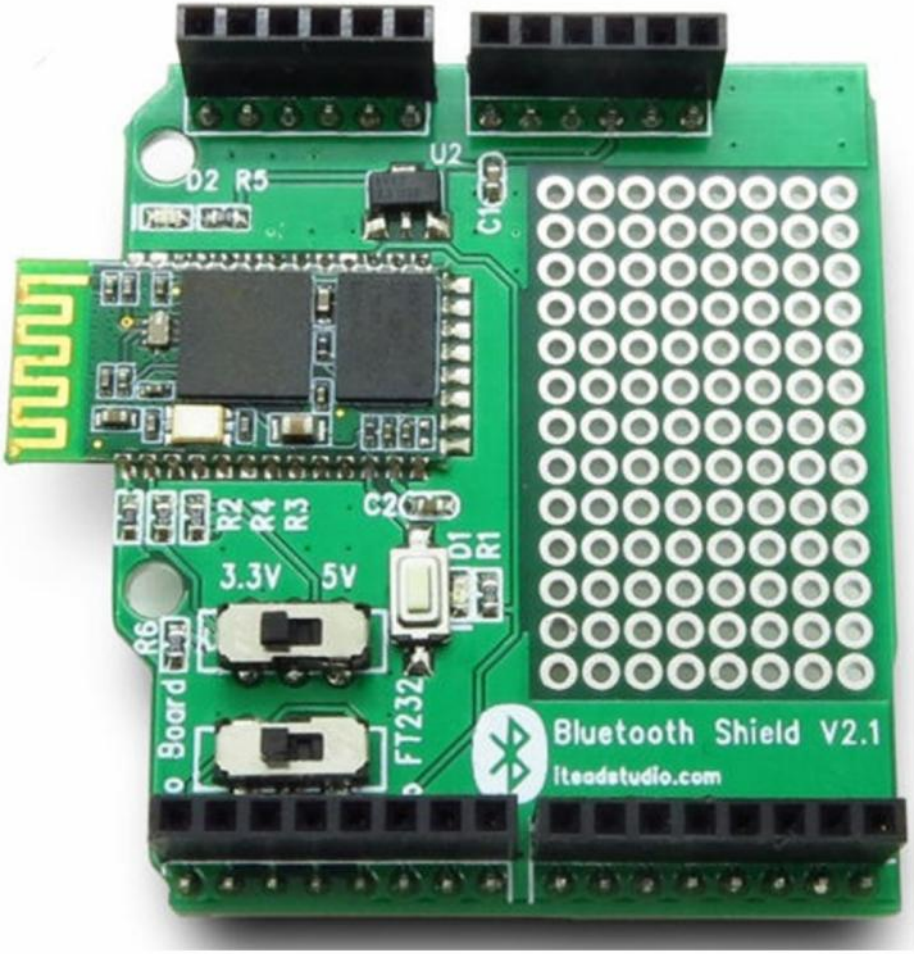
ADXL345 3-Axis Accelerometer GY-291 (5,33€)

5V Geared Stepper Motor 28BYJ-48 ULN2003 Driver (5,30€)

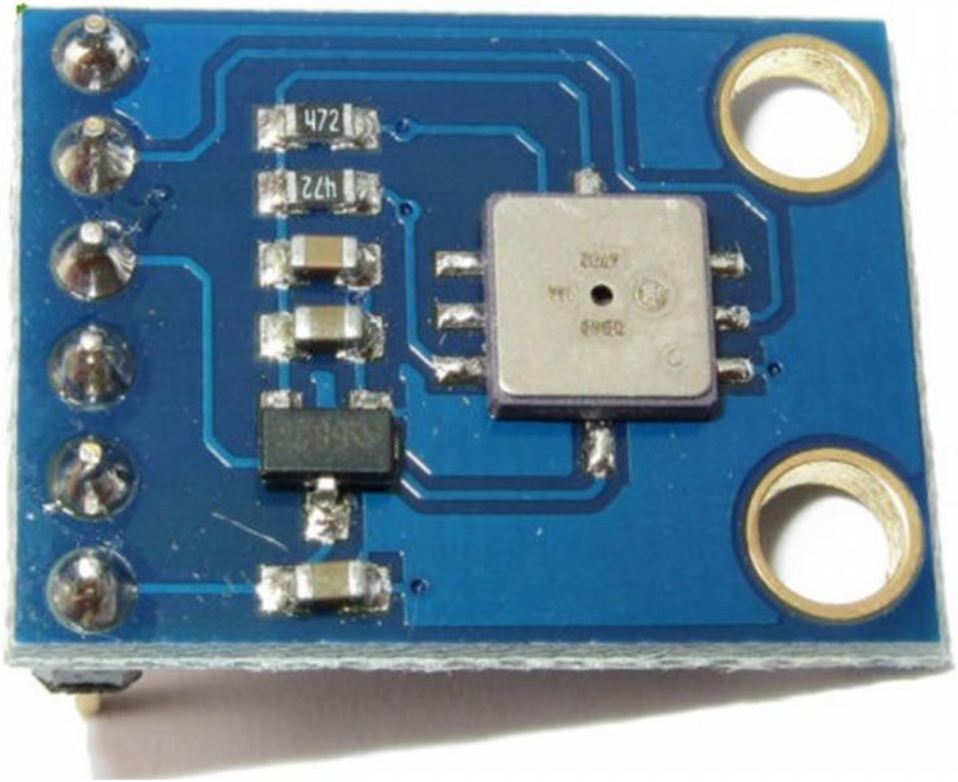
5V Geared Stepper Motor 28BYJ-48 ULN2003 Driver (5,30€)

4 x 4 Membrane 16 Key Keypad (4,04€)

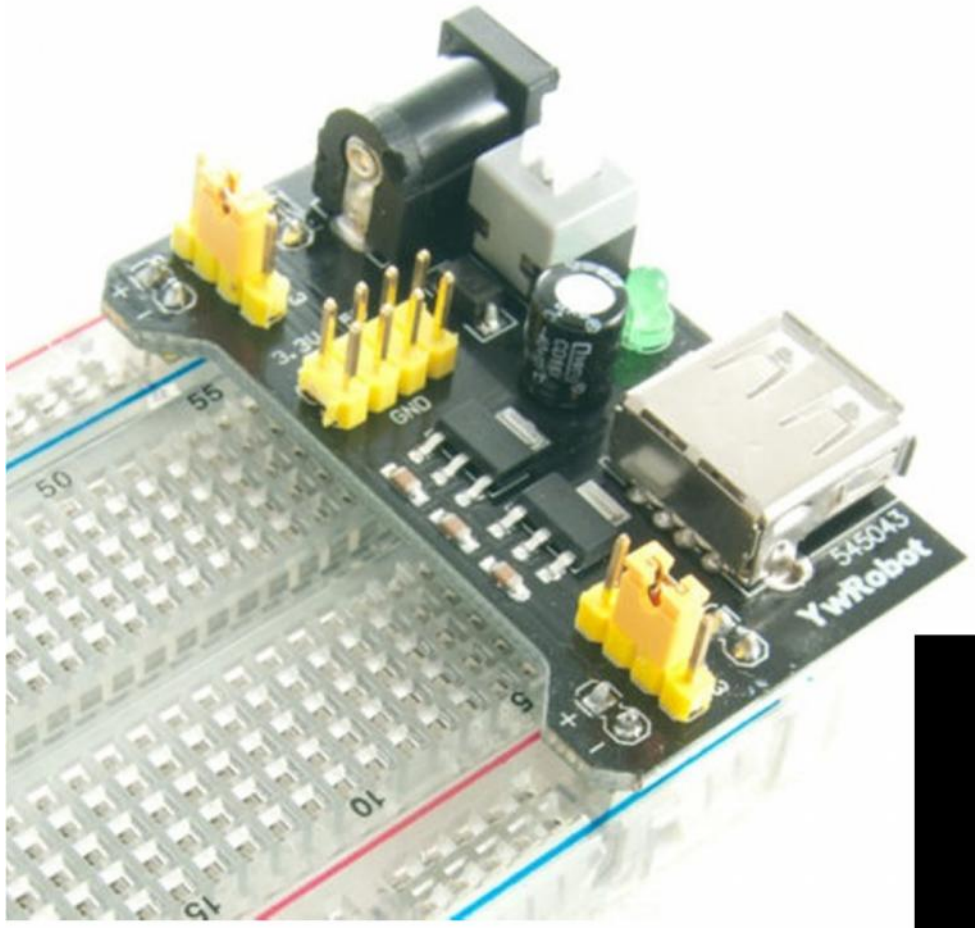
4 x 3 Membrane 12 Key Keypad (3,80€)



Bluetooth Shiled V2.1 (14,14€)

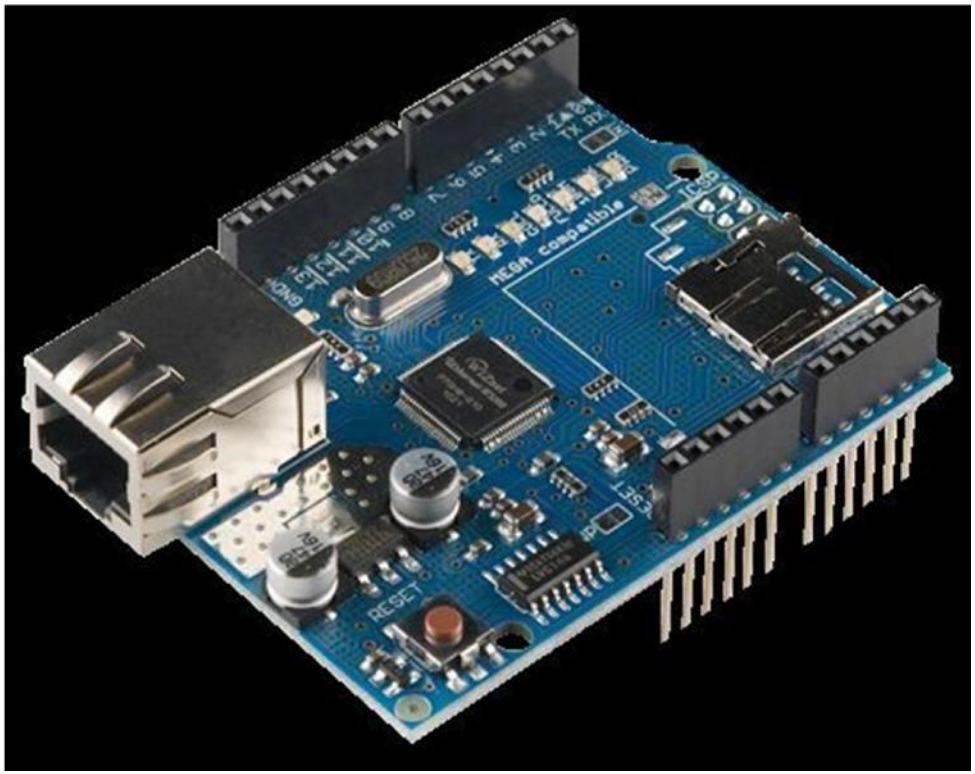


BMP085 Barometric Pressure Temperature Altitude GY-65 (5,18€)

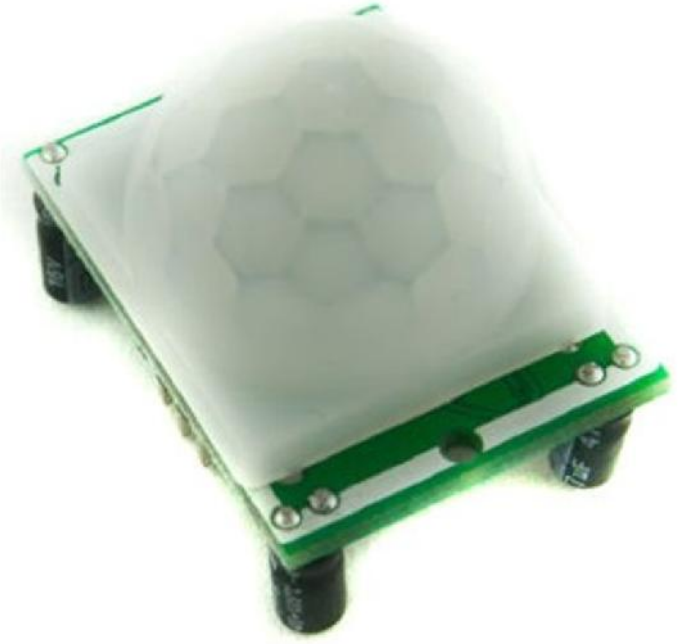


Breadboard Power supply 3.3v or 5v USB or 2.1mm (3,35€)

Ethernet Network Shield for Arduino UNO Mega W5100 (13,77€)

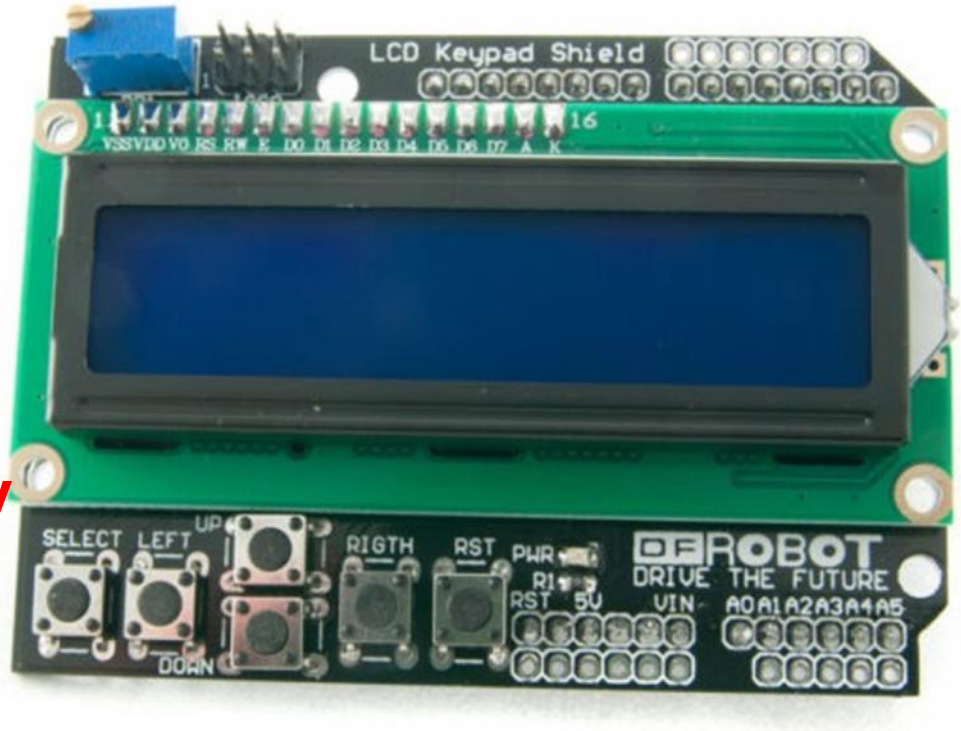


Colorduino RGB LED matrix driver (14,97€)



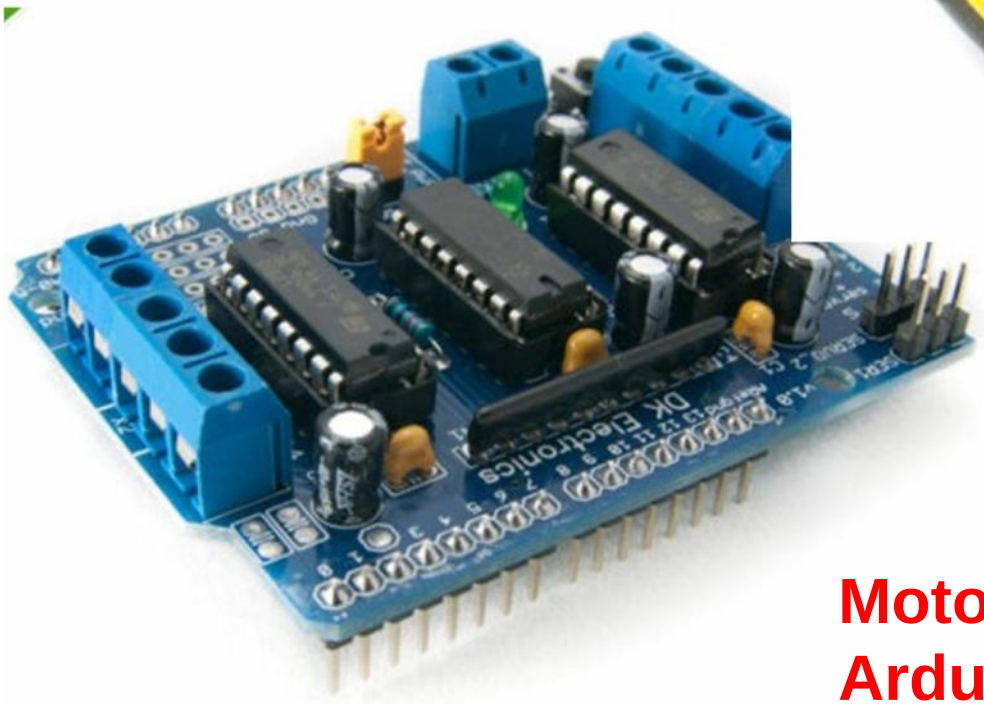
HC-SR501 PIR Infrared Motion Sensor (4,80€)

HR202 Humidity and temperature module (4,67€)



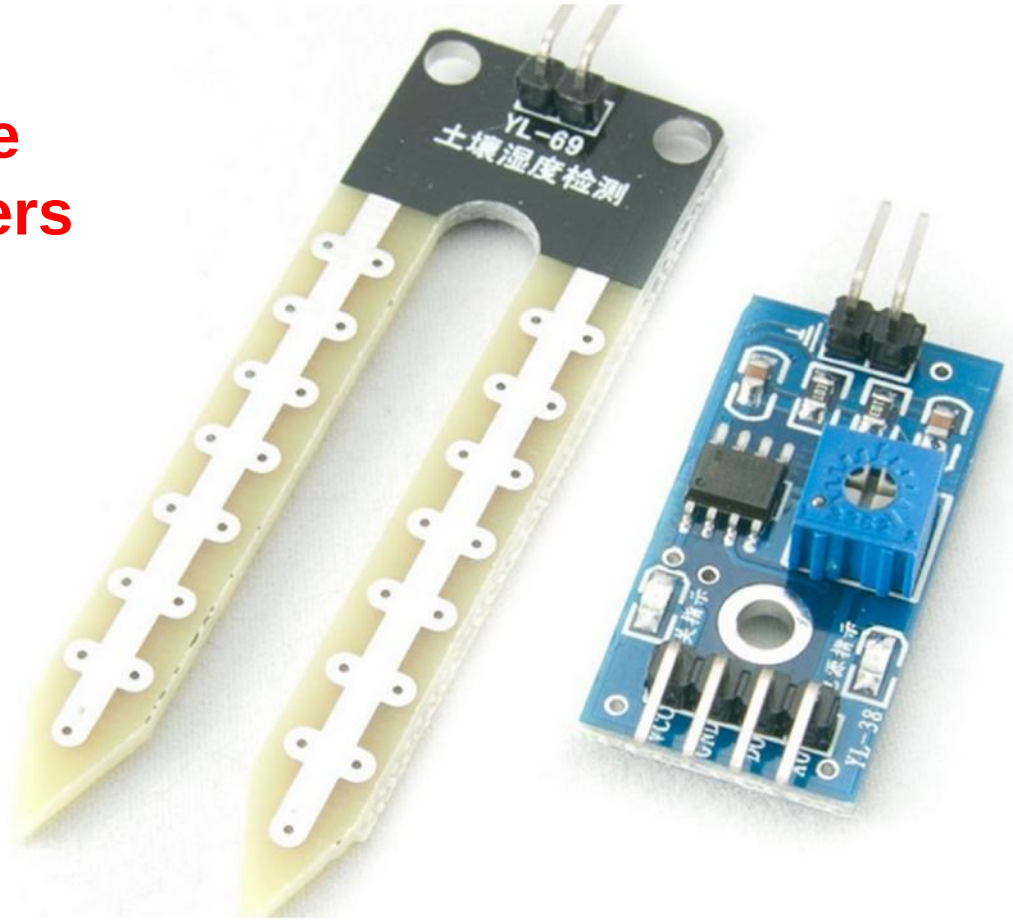
LCD Keypad Shield For Arduino - Blue 2x16 Display (7,57€)

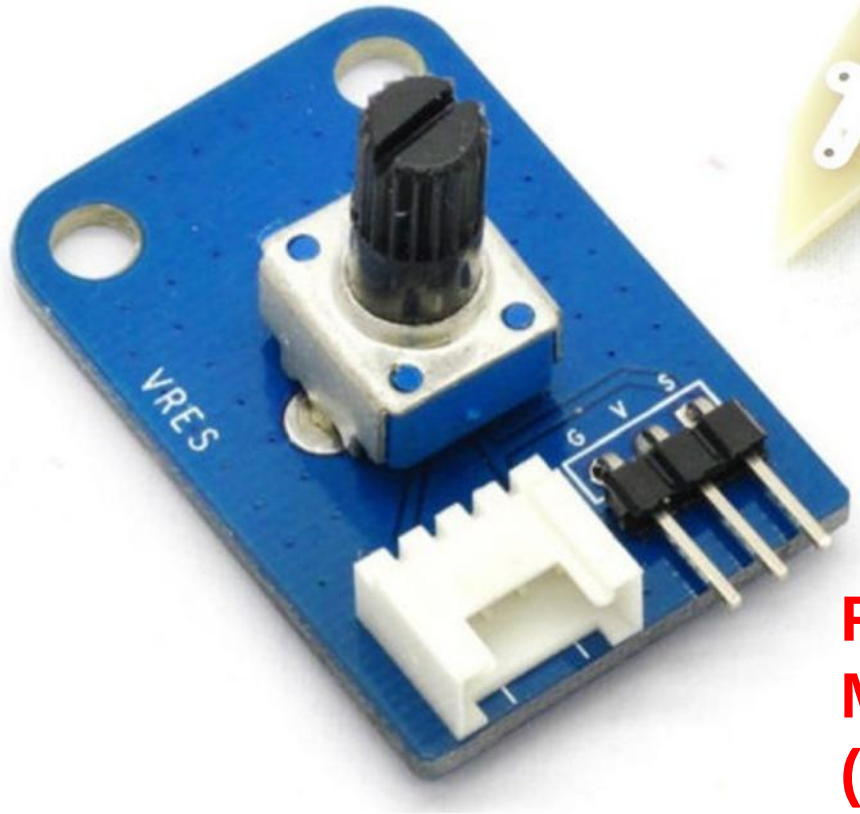
Robot Wheel + Motor + Tyre DC 3V-8V Arduino Smart Car (5,56€)



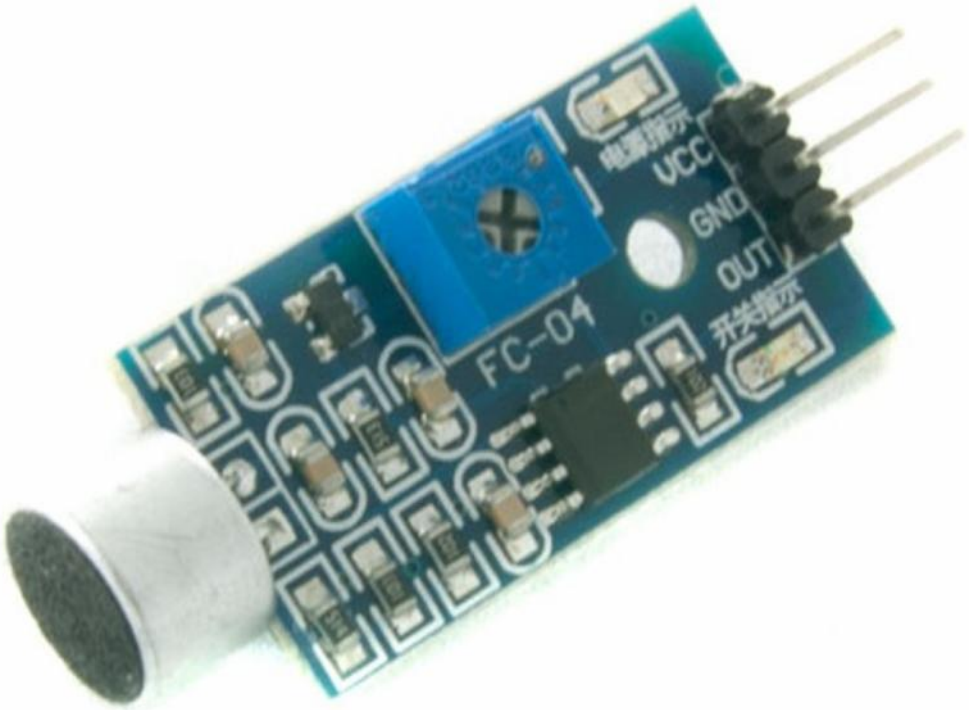
Motor Drive Shield L293D for Arduino Robotics (8,83€)

Soil Hygrometer Moisture Sensor Probe and Jumpers (6,82€)

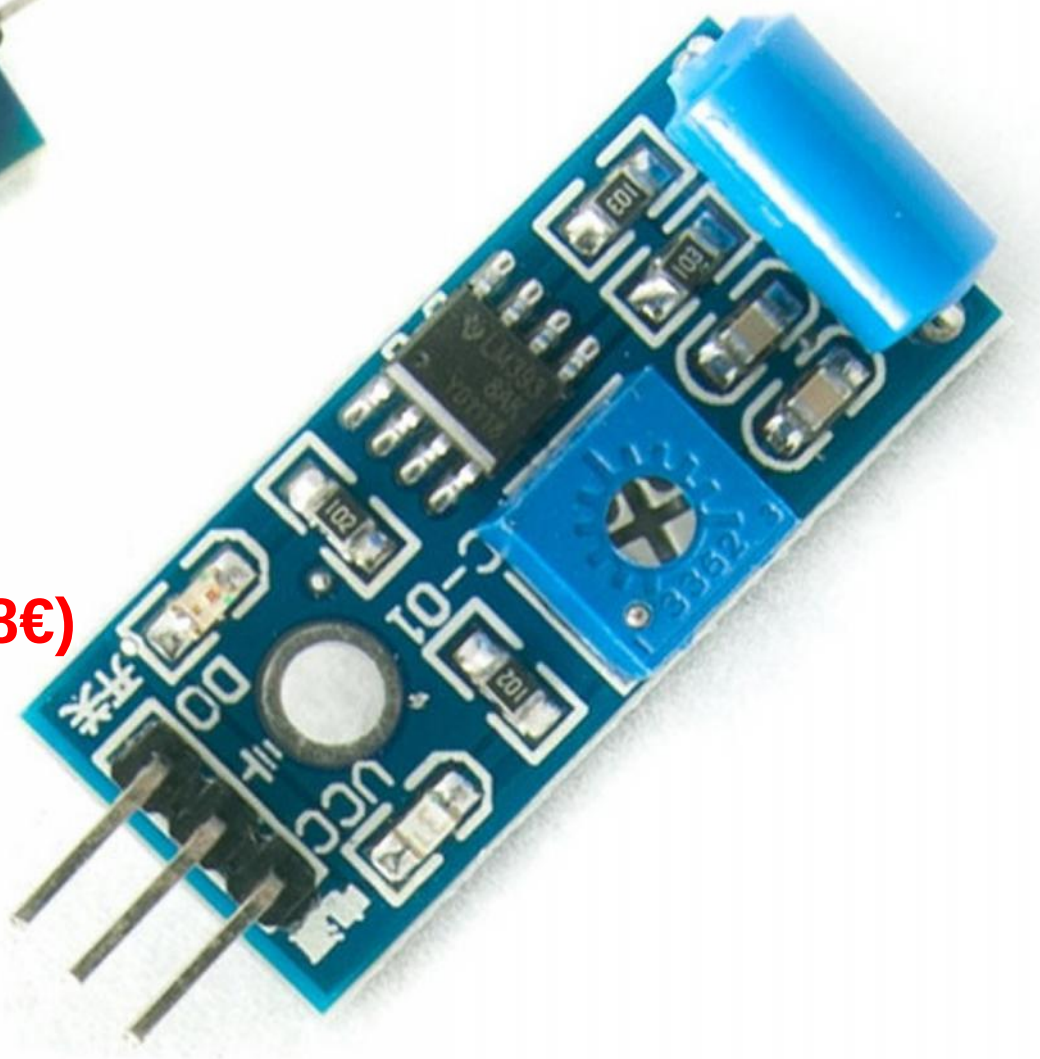




Rotary Potentiometer Brick Module - Rotation sensor (3,79€)



Sound Sensor Module (5.18€)



SW-420 vibration sensor (3,16€)

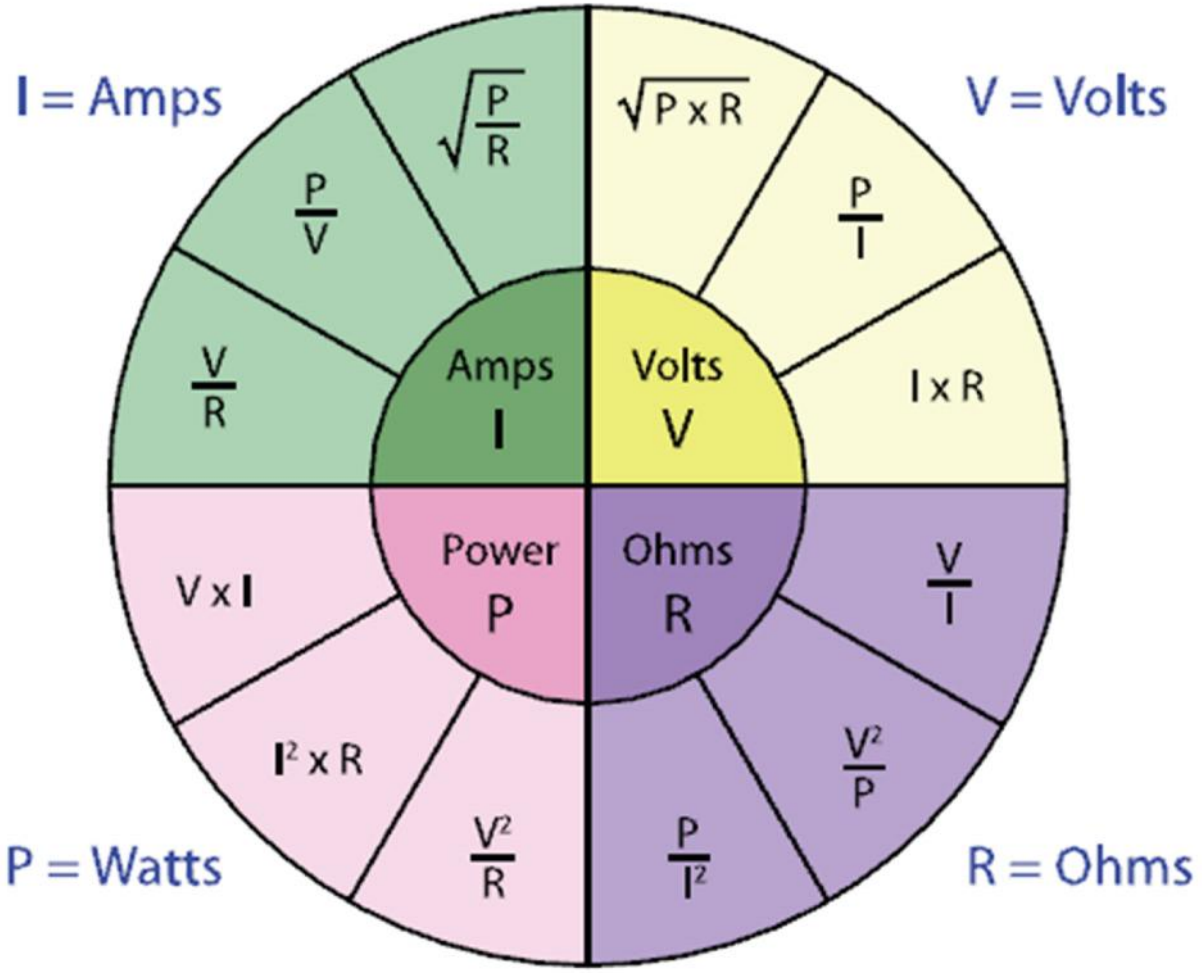
USBASP USB ISP Programmer & Cable & Converter AVR ATMEL (8,34€)

Ultrasonic Range finder_distance sensor HC-SR04 (3,79€)

Che cos'è un SENSORE

Un sensore è un dispositivo di ingresso usato per riconoscere o misurare una grandezza fisica.

Alcuni esempi includono i sensori che percepiscono la luce, la temperatura, la pressione e le sostanze chimiche (come per esempio il monossido di carbonio CO2).



Configurazione sicuramente funzionante

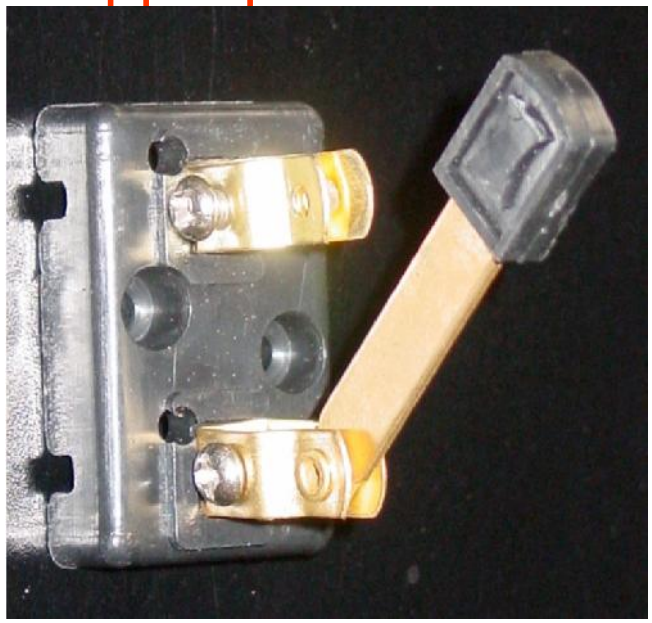
Regola # 1 dello sperimentatore:

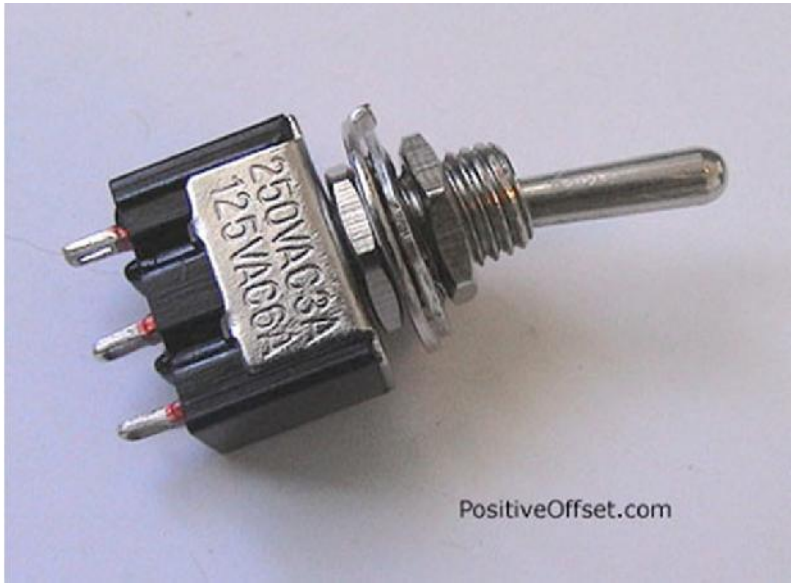
Prima di provare qualcosa di nuovo, partire da una situazione o uno stato sicuramente funzionante sia dell'Hardware che del Software.

Quindi cerchiamo di caricare il software "Blink" in modo che lavori perfettamente.

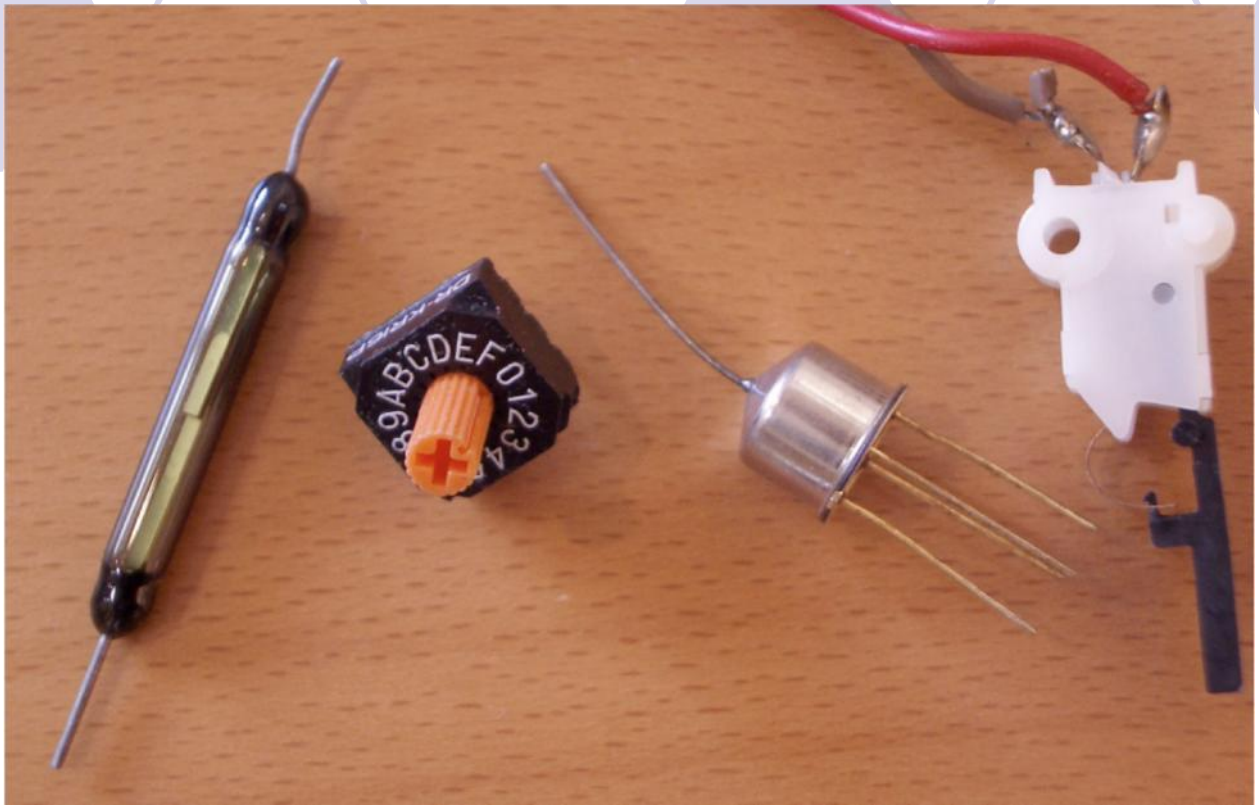
Ingressi digitali

- La maggior parte degli ingressi (digital input) che userete sono interruttori, pulsanti, contatti di fine corsa, ecc.
- Gli interruttori consentono di interrompere o abilitare il passaggio della corrente
- Fondamentalmente, sono tutti come il sezionatore semplice (figura a sinistra)
- Unipolare = un solo cavo viene controllato
  Doppio polo = due cavi vengono controllati in una sola





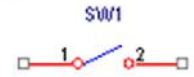
Altri tipi di interruttori e contatti.



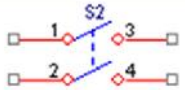
Il **sensore di inclinazione (tilt)** ha una pallina che sente il movimento.
Gli **interruttori reed** (magnetici) si chiudono appena viene avvicinato un piccolo magnete (nella figura il 1° a sinistra).
L'interruttore esadecimale (2° a sinistra) è in realtà un deviatore con molti interruttori in uno.

Interruttori, Pulsanti e Deviatori

L'**interruttore**, dopo il rilascio, memorizza lo stato **APERTO** o **CHIUSO** del suo contatto



INTERRUTTORE
1 CONTATTO



INTERRUTTORE
2 CONTATTI

Il **pulsante**, dopo il rilascio, ritorna nella posizione iniziale che aveva prima della sua pressione. Esistono due differenti tipi:

- Pulsante di tipo **normalmente aperto N.A.**
- Pulsante di tipo **normalmente chiuso N.C.**

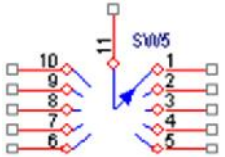
PULSANTE
NORMALMENTE
APERTO

PULSANTE
NORMALMENTE
CHIUSO

Il **deviatore**, dopo l'azionamento, memorizza uno dei differenti percorsi selezionabili.

DEVIATORE 1
CONTATTO 2
POSIZIONI

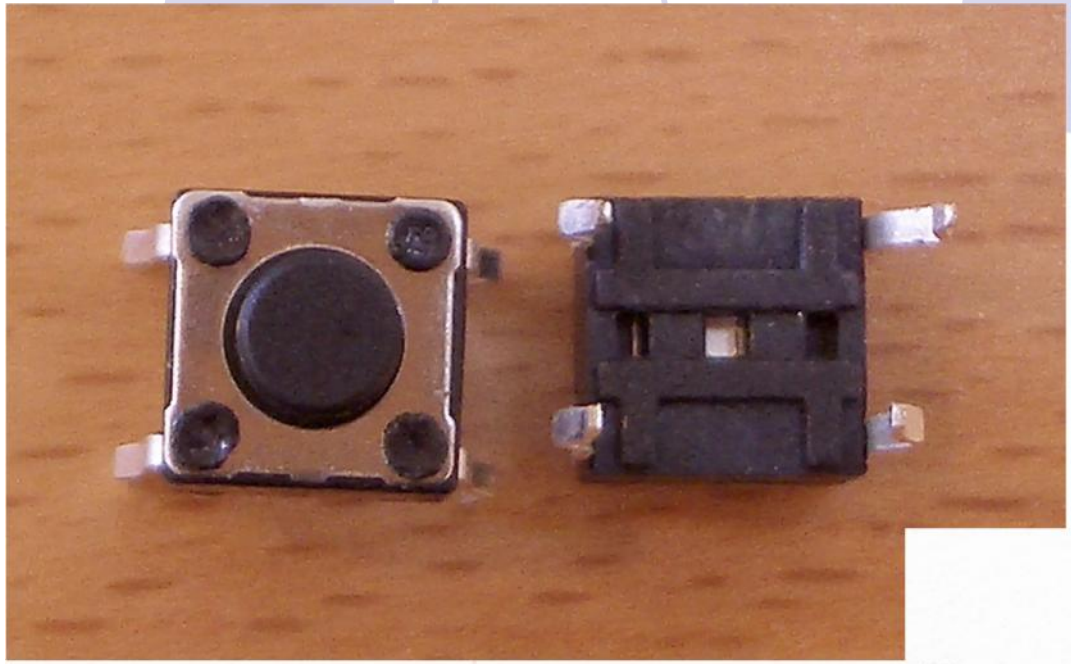
DEVIATORE 2
CONTATTI 2
POSIZIONI

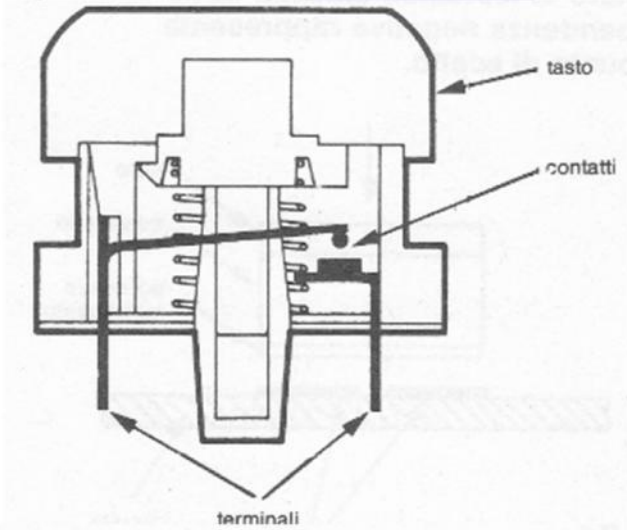


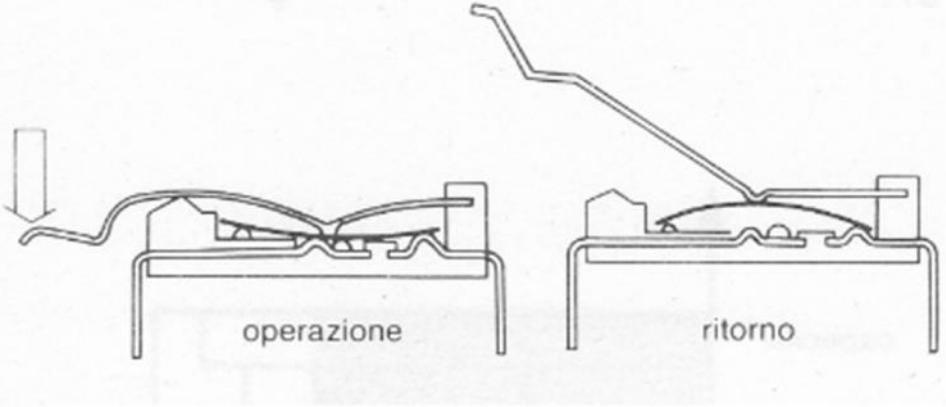
DEVIATORE 1
CONTATTO 10
POSIZIONI

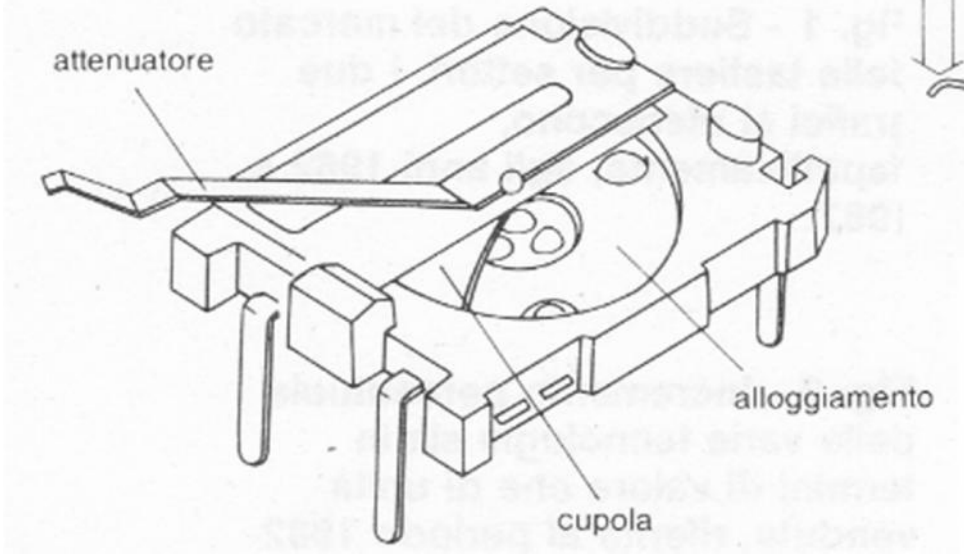
Negli schemi elettrici tutti gli organi in movimento vengono disegnati per convenzione in condizione di **riposo** (senza attivarli)

Pulsanti piccoli da circuito stampato e breadborad









Interruttori e Pulsanti

L'interruttore, deviatore o pulsante permettono il passaggio o l'interruzione della corrente.

Ma Arduino ha bisogno di "leggere" una tensione:

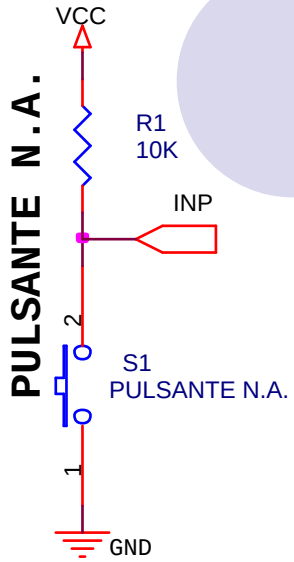
1) Un livello logico **alto = HIGH = +5V = VCC**

2) Un livello logico **basso = LOW = 0V = GND**

L'interruttore e il pulsante si definiscono **chiusi** (resistenza tra i suoi due terminali **< 1 ohm = cortocircuito**), quando consentono il passaggio di corrente, invece se il passaggio è interdetto si definiscono **aperti** (resistenza **> 10 Mohm**)

Collegamento dei pulsanti N.A.

(normalmente aperti)

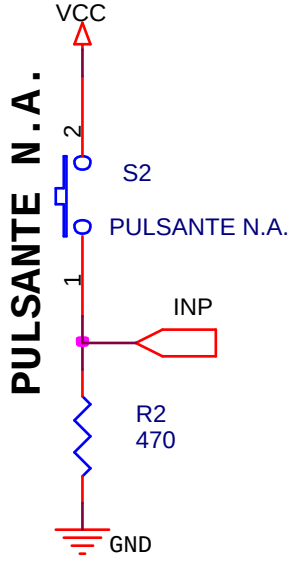


Circuito con resistenza di **pull-up** per collegare un pulsante di tipo **N.O.** (***normaly open***) a un pin del microcontrollore.

Pulsante **premuto** → livello logico in uscita **0**

Pulsante **rilasciato** → livello logico in uscita **1**

```
PULS. PREMUTO = LOW
PULS. NON PREMUTO = HIGH
```



Circuito con resistenza di **pull-down** per collegare un pulsante di tipo **N.O.** (***normaly open***) a un pin del microcontrollore.

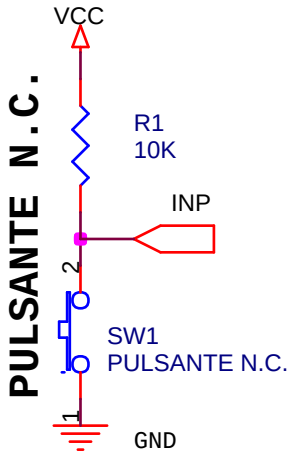
Pulsante **premuto** → livello logico in uscita **1**

Pulsante **rilasciato** → livello logico in uscita **0**

```
PULS. PREMUTO = HIGH
PULS. NON PREMUTO = LOW
```

Collegamento dei pulsanti N.C.

(normalmente chiusi)

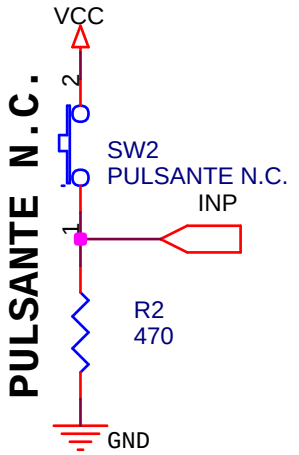


Circuito con resistenza di **pull-up** per collegare un pulsante di tipo **N.C.** (***normaly close***) a un pin del microcontrollore.

Pulsante **premuto** → livello logico in uscita **1**

Pulsante **rilasciato** → livello logico in uscita **0**

```
PULS. PREMUTO = HIGH
PULS. NON PREMUTO = LOW
```



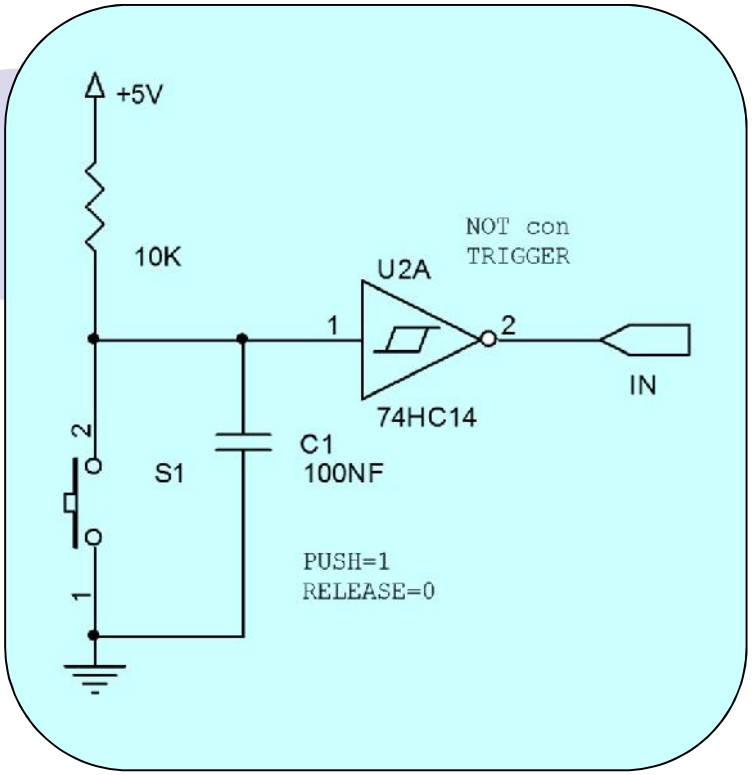
Circuito con resistenza di **pull-down** per collegare un pulsante di tipo **N.C.** (***normaly close***) a un pin del microcontrollore.

Pulsante **premuto** → livello logico in uscita **0**

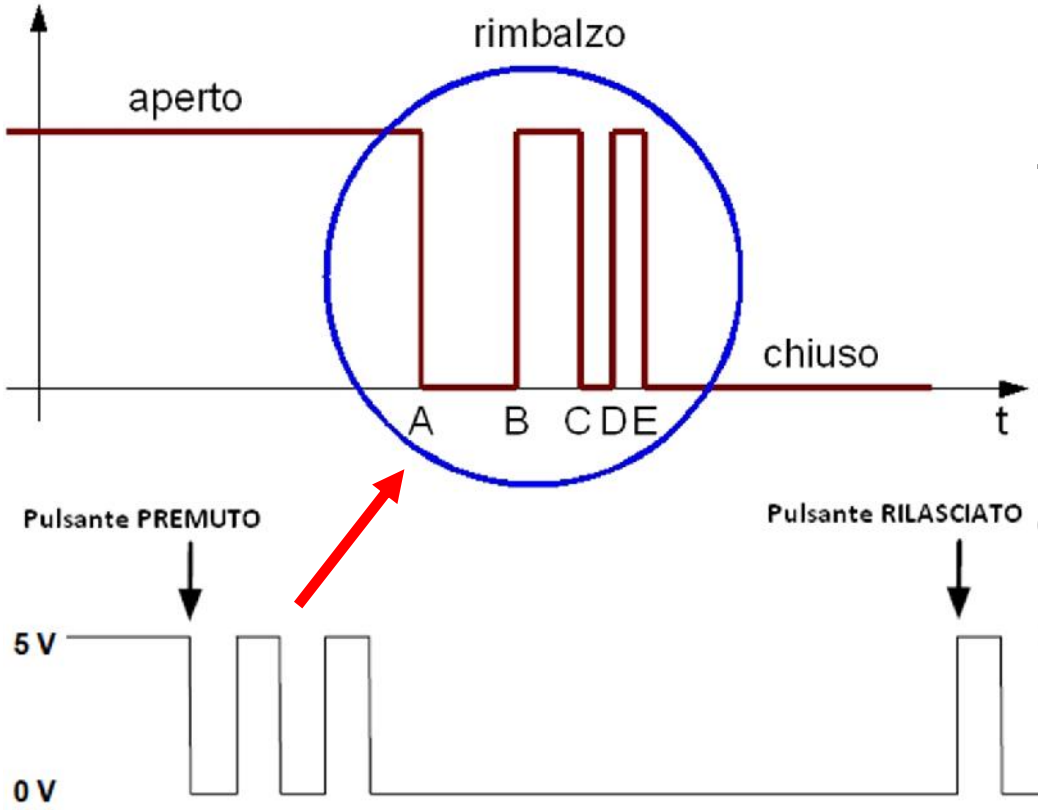
Pulsante **rilasciato** → livello logico in uscita **1**

```
PULS. PREMUTO = LOW
PULS. NON PREMUTO = HIGH
```

Circuito elimina rimbalzi (antibounce)



Circuito con porta NOT a trigger per eliminare a livello hardware i tipici rimbalzi dei contatti di un pulsante in chiusura della durata di 1÷10ms.



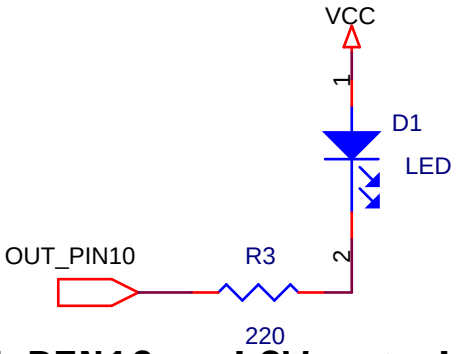
Tipici rimbalzi dei contatti di un pulsante in chiusura della durata di 1÷10ms.

E' possibile anche eliminare i rimbalzi dei contatti con un apposito software.

Accensione dei led con Arduino

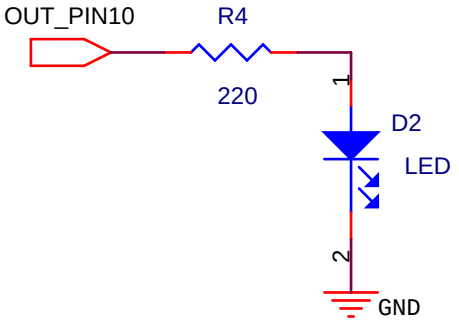
- Ogni pin è in grado di fornire circa **40 mA** (15mA → Arduino DUE) di corrente, questa corrente è sufficiente per lavorare con un diodo LED (**max. 20 mA**). Valori assorbiti o erogati che sono superiori ai **40 mA** o tensioni superiori a **5V** (**3,3V** → Arduino DUE) su qualsiasi pin possono danneggiare il microcontrollore o il dispositivo collegato.

Led acceso con un livello **LOW**




```
OUT_PIN10 = LOW --> LED ACCESO
OUT_PIN10 = HIGH --> LED SPENTO
```

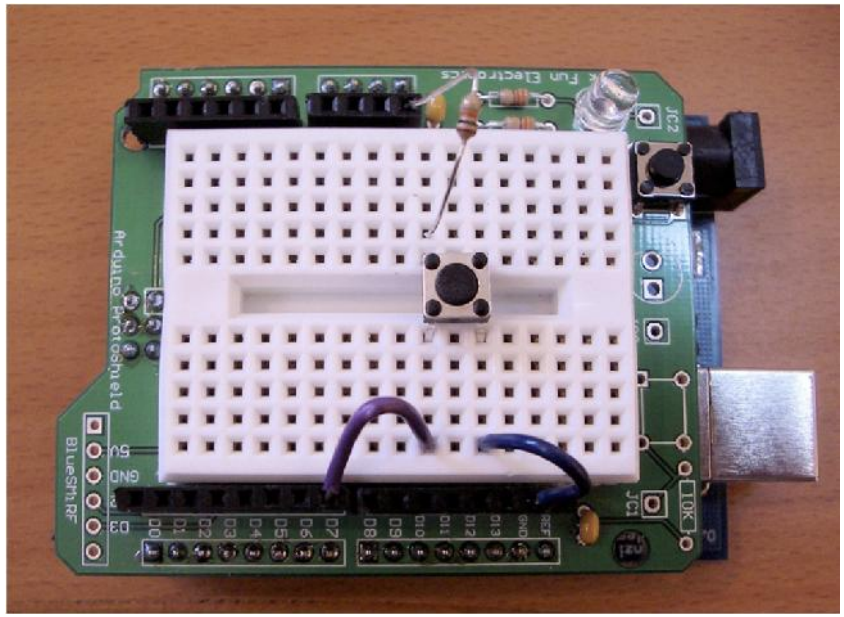
Led acceso con un livello **HIGH**

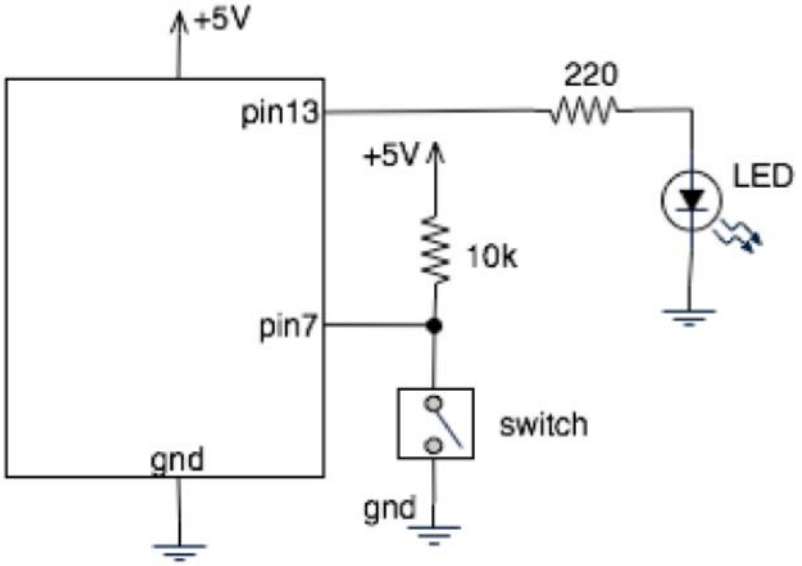



```
OUT_PIN10 = HIGH --> LED ACCESO
OUT_PIN10 = LOW --> LED SPENTO
```

Arduino Uno con l'input / output digitale

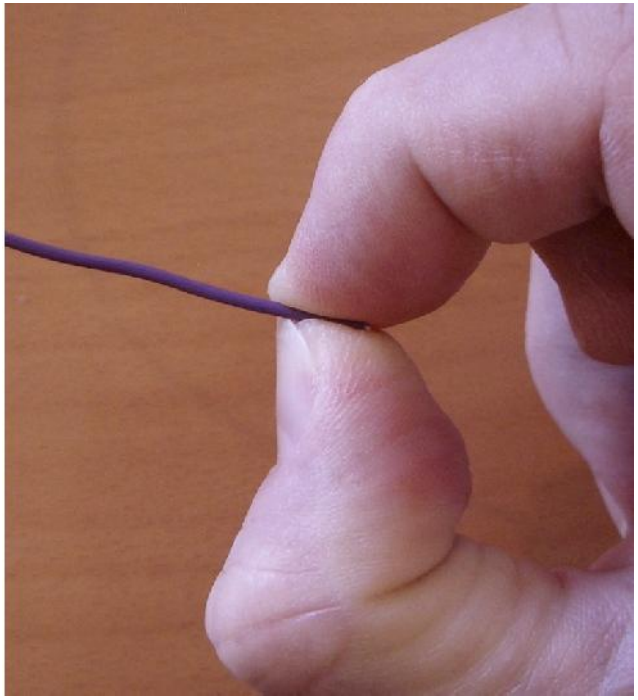
- Come **INPUT** è possibile collegare e configurare qualsiasi pulsante o interruttore tra i pin **2** e **12** della scheda [sono da escludere i pin **0** (**RX**), **1** (**TX**) e **13** (**led interno**)]
- Come **OUTPUT** è possibile collegare e configurare qualsiasi led tra i pin **2** e **13** della scheda [sono da escludere i pin **0** (**RX**), **1** (**TX**)]

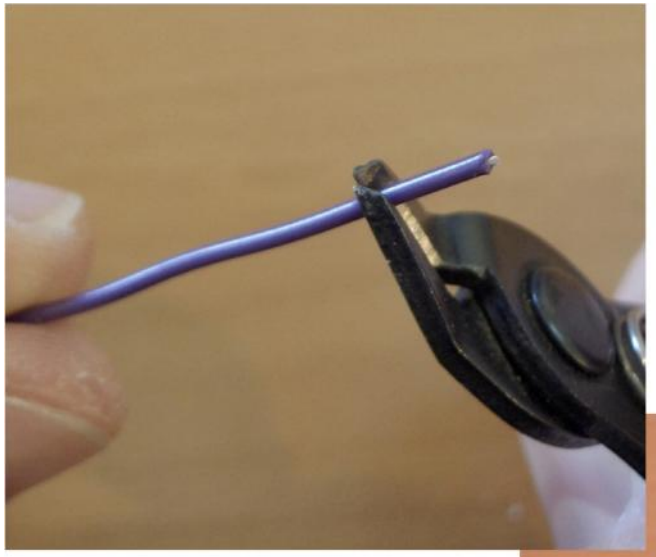


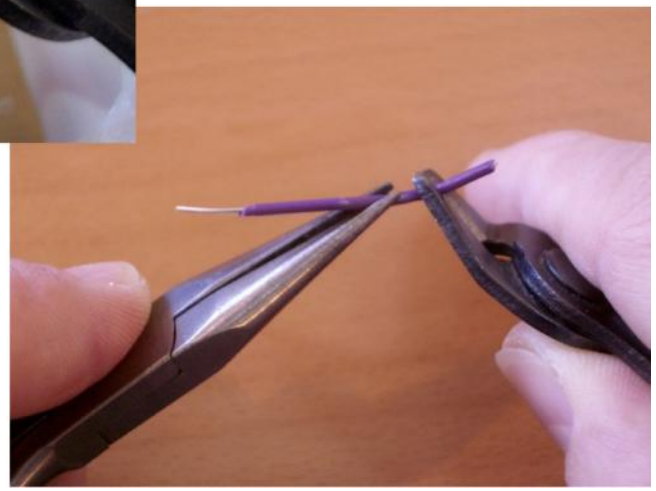


Come effettuare i collegamenti con il cavo

- Tagliare la lunghezza del cavo necessaria
- Spelare con le forbici da elettricista o con lo spellafili entrambe le estremità per 1 cm massimo.
- Non utilizzare i denti per spelare i cavi







Come effettuare i collegamenti con il cavo

- Il risultato finale

- Confezione pronta (sconsigliata perché è da acquistare)

SOFTWARE

Utilizzo della funzione **setup()** e **loop()**

- **setup()** è la funzione per l'inizializzazione degli input e output. Viene eseguita solo una volta, ed è usata per impostare le modalità di funzionamento dei pin come input/output (pinMode) o per inizializzare la comunicazione seriale.
- **loop()** è la funzione principale per l'esecuzione. Include il codice (sequenza di istruzioni) che deve essere eseguito in un ciclo infinito (loop)
- Entrambe le funzioni sono indispensabili per il programma di lavoro (sketch)

- Le **parentesi graffe** si scrivono con **Alt+123** → "**{**" e **Alt+125** → "**}**" sul tastierino numerico della tastiera.

Utilizzo della funzione digitalRead() e pinMode()

In **setup()** utilizzare **pinMode(numero_pin, INPUT);**

- **numero_pin** = fornire il numero del pin da utilizzare come input oppure come output

es.: **pinMode(7, INPUT);** // definisci il pin 7 come input

pinMode(8, OUTPUT); // definisci il pin 8 come output

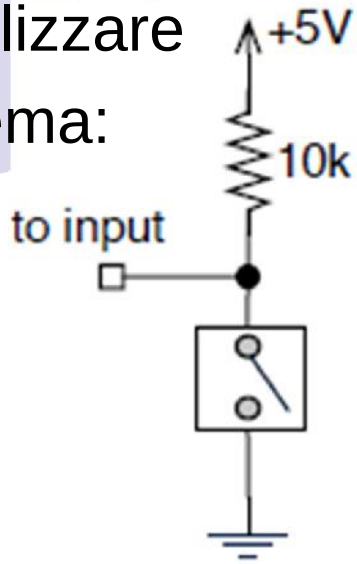
In **loop()** utilizzare **digitalRead(numero_pin);** per ottenere il livello logico acquisito sull'input (pulsante, interruttore, ecc.)

se necessario il valore letto può essere memorizzato in una variabile.

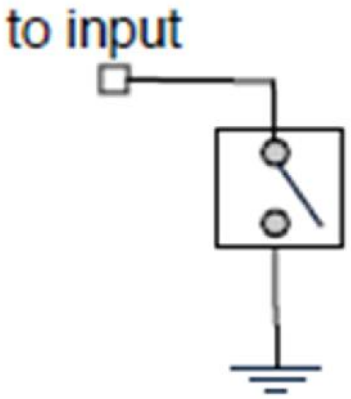
es.: **leggi_pulsante = digitalRead(7);** // leggi il valore dall'input collegato al pin7 (i valori sono "0" oppure "1") e memorizzalo nella variabile denominata «leggi_pulsante»

Pulsanti e interruttori senza Resistori di pull-up esterni

Invece di utilizzare questo schema:



Potete lavorare con questo:



ATTENZIONE!
Non esiste la resistenza di pull-down all'interno del micro, solo quella di pull-up.

Ma come si effettua la programmazione delle resistenze interne di pull-up?

```
int pin_pulsante = 7; // pin 7 collegato al pulsante N.A.

void setup() // funzione di inizializzazione dei INPUT/OUTPUT
{
  pinMode(pin_pulsante, INPUT); // inizializza il pin 7 come INPUT collegato al pulsante n.a.
  digitalWrite(pin_pulsante, HIGH); // attiva sul pin 7 la resistenza da 10Kohm di pullup
}
```

SOFTWARE - Comunicare con gli altri

Arduino può utilizzare lo stesso cavo USB utilizzato per la programmazione per comunicare con i computer.

Serial.begin(); – predispone i parametri della seriale USB (RS232 fittizia)

Serial.print(); – per inviare dei dati al computer (USB)

Serial.write(); – per inviare dei dati in codice ASCII al computer (USB)

Serial.read(); - per leggere i dati inviati dal computer (USB)

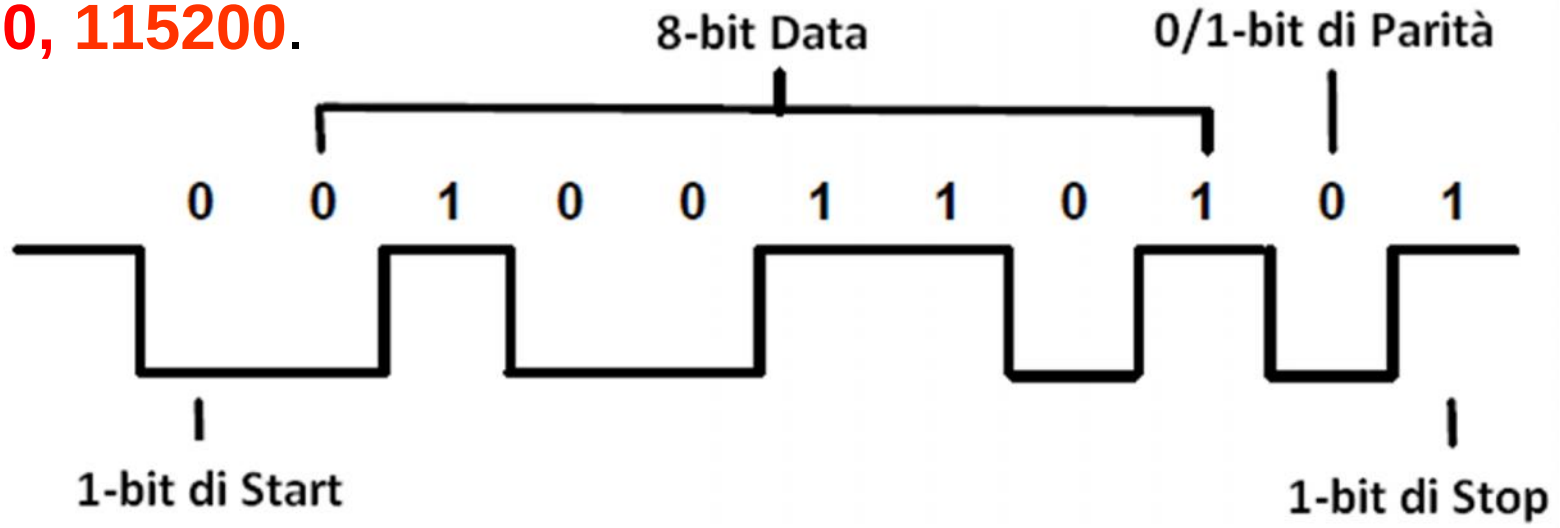
Utilizzo della funzione Serial.print()

La funzione “**Serial.print();**” trasferisce (stampa) i dati sulla porta seriale RS232 virtuale (USB reale).

La funzione “**Serial.println();**”. È simile alla precedente con l’aggiunta di un ritorno automatico a capo e avanzamento di riga.

Per configurare la porta seriale **RS232** e impostare il baud rate (velocità di trasmissione dei caratteri) si utilizza dentro il **setup()** la funzione **Serial.begin(9600);”.**

Il valore tipico di trasmissione e ricezione per comunicare con il computer è di **9600 baud con 1-bit di Start, 8-bit di Data 0/1-bit parità e 1-bit di Stop**. Velocità maggiori sono supportate **19200, 38400, 115200**.



Invio dati al Computer

Serial.print(), Serial.println() e Serial.write()

int test = 33; // valore numerico coincide con carattere "**!**"

Serial.print(test); // stampa i caratteri ascii "**33**". Di default è il valore DECIMALE

Serial.write(test); // stampa il carattere ASCII "**!**".

Serial.print(test, DEC); // stampa i caratteri "**33**".

Serial.print(test, HEX); // stampa i caratteri "**21**". Valore in esadecimale (base 16)

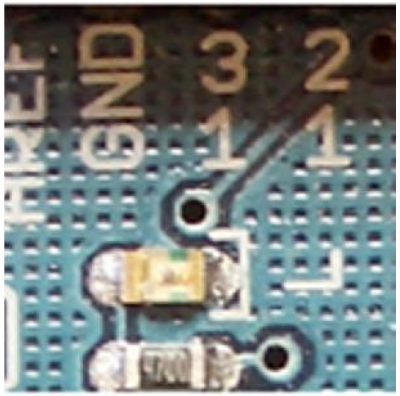
Serial.print(test, OCT); // stampa i caratteri "**41**". Valore in ottale (base 8);

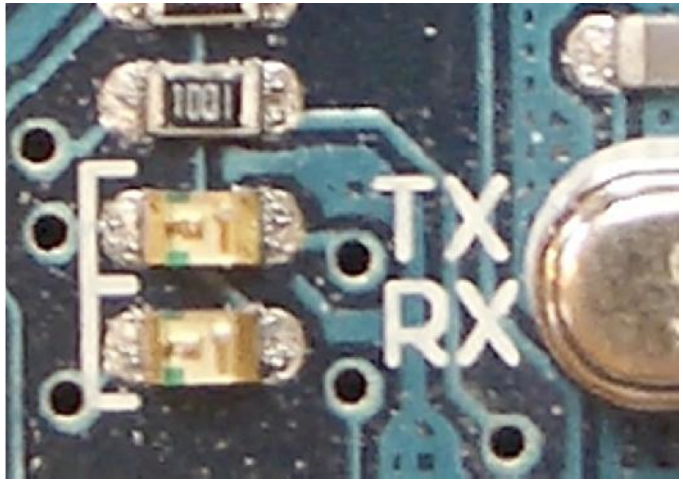
Serial.print(test, BIN); // stampa i caratteri "**100001**". Valore in binario (base 2)

Stesse modalità con la funzione "**Serial.println()**" con il cursore che salta su una nuova riga a capo.

Comunicazione seriale
Guardiamo i led **TX / RX**

- **TX** - invio dati al PC
- **RX** – ricezione dati dal PC
- Questi due led vengono usati durante la programmazione per la comunicazione USB





Comunicazione seriale

- "**Seriale**" perché i dati vengono suddivisi in parecchi bit, ognuno dei quali viene inviato in tempi successivi, cioè uno dopo l'altro su un singolo cavo.

- Solo un cavo dei dati è necessario per inviare e un secondo cavo per ricevere.
- Nota → in realtà occorre anche utilizzare un cavo di ritorno o di massa GND per permettere alla corrente del generatore di confluire allo stesso.

'H'

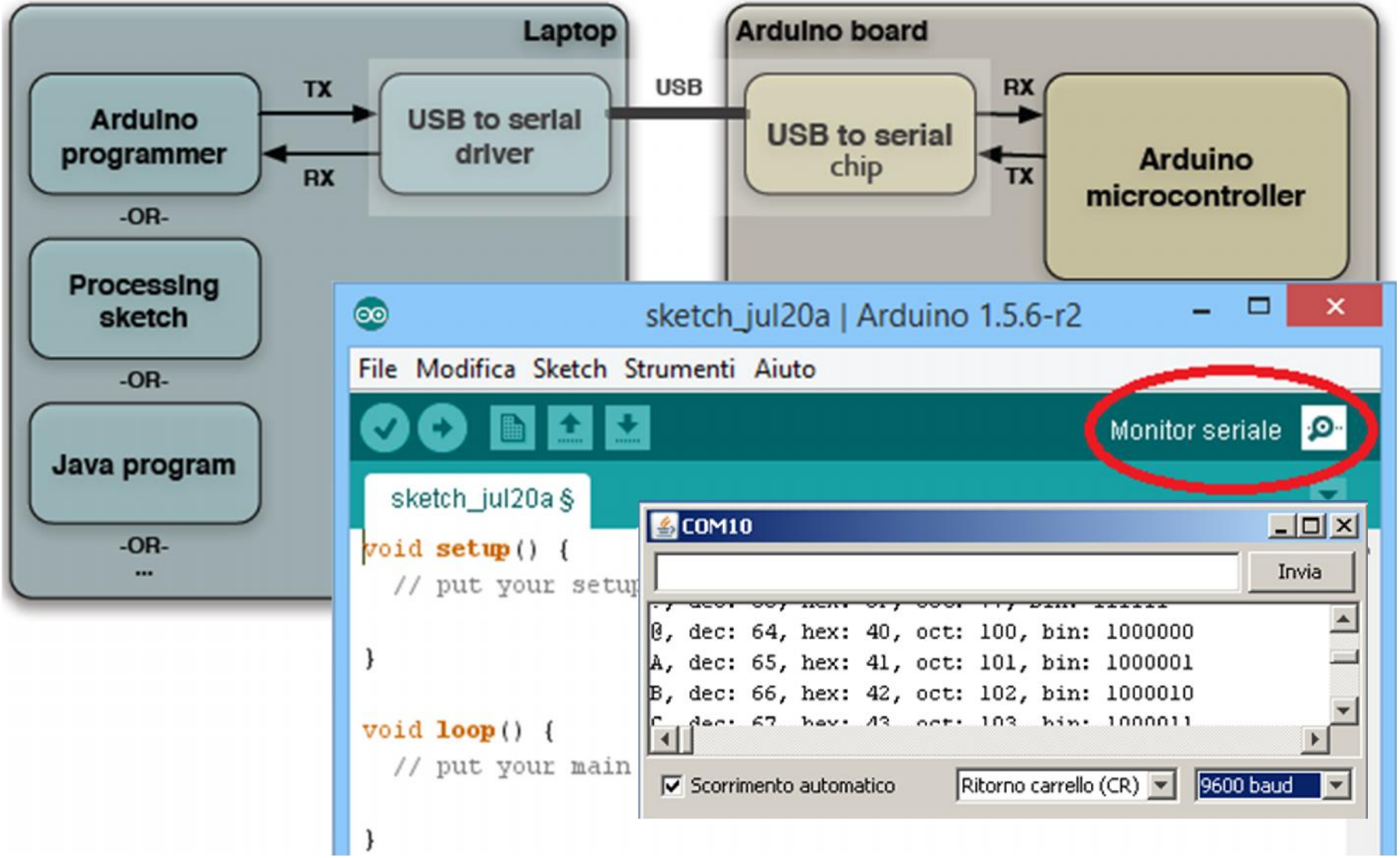
= 0 1 0 0 1 0 0 0

= L H L L H L L L

=

Arduino ← → Computer

- L’interfaccia USB per Arduino rende le comunicazioni più semplici. I computer attuali non possiedono più la vecchia e obsoleta interfaccia RS232.



Arduino & USB

- La scheda Arduino UNO non contiene tutto di serie perché l'USB implementata è solo di tipo "host" quindi non risulta possibile gestire un interfacciamento a unità flash USB, hard disk USB, webcam USB, ecc.., a causa delle modeste capacità di elaborazione del microcontrollore.
- Con le nuove schede Arduino **DUE** e Arduino **YUN** le precedenti limitazioni vengono eliminate.

SOFTWARE
Le istruzioni

- Le ***istruzioni*** *nel linguaggio C* esprimono *azioni* che, una volta eseguite, comportano una *modifica permanente dello stato interno* del programma o del mondo circostante.
- Le ***strutture di controllo*** permettono di aggregare istruzioni semplici in istruzioni più complesse.

Tipi di istruzioni che utilizzeremo:

- **if …. else**
- **while()**
- **do …. while()**
- **for**
- **switch …. case**

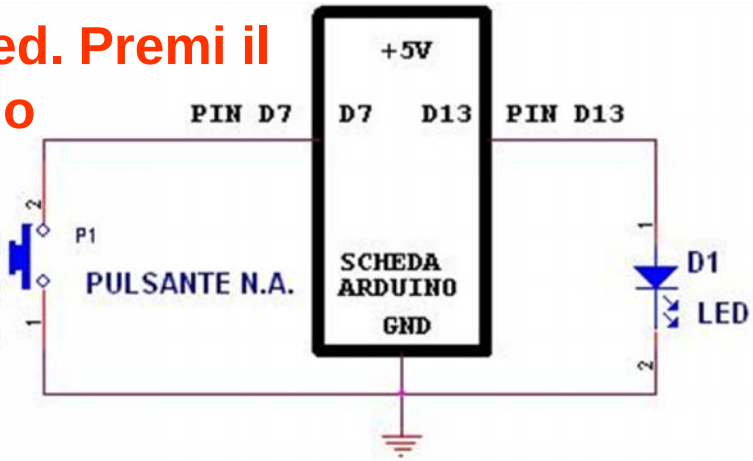
Utilizzo della istruzione if() …. else ….

- L'istruzione «**if()**» controlla se la condizione tra le parentesi tonde risulta «**VERA**», esegue la sequenza di istruzioni comprese tra le prime parentesi graffe, mentre se la condizione è «**FALSA**» esegue la sequenza di istruzioni disponibile dopo la parola «else» e comunque delimitata dalle parentesi graffe aperta e chiusa.
- È possibile trovare una istruzione «if()» senza il corrispondente «else», mentre non risulta possibile trovare un «else» senza il proprio «if».
- Se la condizione VERA dell'if oppure la condizione FALSA dell'else è comprensiva di una sola istruzione è possibile eliminare le parentesi graffe.

```
int a = 10, b = 20; // assegna alla variabile "a" il valore 10 e alla "b" il valore 20
if (a == 10) // se e' VERA la condizione che la variabile "a" vale 10
  {          // esegui la prossima istruzione
a = a + b; // dopo l'istruzione la variabile "a" vale 30
  }
else  // se la condizione è falsa esegui la prossima istruzione
  {
     a = a - b; // dopo l'istruzione la variabile "a" vale -10
  }
```

```
int a = 10, b = 20; // assegna alla variabile "a" il valore 10 e alla "b" il valore 20
if (b > 30) // se la condizione (20 > 30) e' VERA (no!!) la condizione che la variabile
            // "b" piu' di 30 esegui la prossima istruzione
   b = b + a; // dopo l'istruzione la variabile "b" vale 30
else  // se la condizione (20 > 30) è FALSA  esegui la prossima istruzione
   b = b - a; // dopo l'istruzione la variabile "b" vale 10
```

Ora è possibile controllare l'accensione del led. Premi il pulsante per accendere, rilascia per spegnerlo




```
/*  I.I.S. Primo LEVI - Torino
  Esercizio N. 3            Data: 03/12/2010
  Progetto: DigitalReadSerial_1       Autore: Questo è un esempio di
  Descrizione: Lettura di un input digitale (pulsante collegato al
  con stampa del livello logico sulla porta seriale e ripetizione de
void setup()    // funzione di inizializzazione della seriale RS232
{
  pinMode(7, INPUT);    // inizializza il pin 7 della scheda Arduino come INPUT (PULSANTE)
  digitalWrite(7, HIGH);  // settaggio per la resistenza interna di pull-up da 10Kohm
  pinMode(13, OUTPUT); // inizializza il pin 13 della scheda Arduino come OUTPUT (LED)
  Serial.begin(9600);  // inizializza la seriale RS232 con 9600 baud, 8 bit dati, nessuna parità e 1 bit di stop
}
void loop()     // programma principale (main) --> ciclo infinito (loop)
{
  int pulsante = digitalRead(7);  // acquisisci il valore dell'input pin 7 nella variabile "pulsante"
  if (pulsante == 0) // verifica se il pulsante è premuto (condizione VERA = pulsante n.a. PREMUTO)
    {
     Serial.print("Pulsante PREMUTO collegato al pin 7 --> Livello:");
     Serial.println(pulsante, DEC); // stampa sulla seriale il valore dell'input collegato al pulsante (pin 7)
     digitalWrite(13, HIGH);    // accendi il LED forzando un livello ALTO sul pin 13
    }
  else  // altrimenti se il pulsante non è premuto (condizione FALSA = pulsante n.a. NON PREMUTO)
    {
     Serial.print("Pulsante NON PREMUTO collegato al pin 7 --> Livello: ");  // stampa sulla seriale
     Serial.println(pulsante, DEC); // stampa sulla seriale il valore dell'input collegato al pulsante (pin 7)
     digitalWrite(13, LOW);     // spegni il LED forzando un livello BASSO sul pin 13
    }
}
```

DigitalReadSerial_1.ino

Operatori di confronto

- I confronti tra due variabili o costanti sono spesso utilizzati nelle istruzioni «**if() … else**», **while()**, ecc. per verificare **se una condizione specificata** è **vera** o **falsa**.
  Le operazioni di confronto utilizzate sono:

- **x == y** → **x è uguale** a **y** (**confronto**)
- **x != y** → **x non è uguale** a **y** (**diverso**)
- **x < y** → **x è minore** di **y**
- **x > y** → **x è maggiore** di **y**
- **x <= y** → **x è minore o uguale** a **y**
- **x >= y** → **x è maggiore o uguale** a **y**

Operatori abbreviati «Compound»

Operatore	Esempio	Espressione equivalente
+=	test += 5;	test = test + 5; // aggiungi 5 alla variabile “test”
-=	test -= 4;	test = test - 4; // sottrai 4 dalla variabile “test”
*=	test *= 3;	test = test * 3; // moltiplica la variabile “test” per 3
/=	test /= 2;	test = test / 2; // dividi la variabile “test” per 2
%=	test %= 2;	test = test % 2; // restituisci il resto della divisione tra la variabile “test” e il valore 2
>>=	test >>= 2;	test = test >> 2; // sposta verso destra di 2 bit la variabile “test”
<<=	test <<= 2;	test = test << 2; // sposta verso sinistra di 2 bit la variabile “test”
&=	test &= 3;	test = test & 3; // esegui l’AND della variabile “test” con il valore 3
|=	test |= 4;	test = test | 4; // esegui l’OR della variabile “test” con il valore 4

Utilizzo della funzione delay()

- Mette in pausa un programma per la quantità di tempo specificato in millisecondi, ad esempio dove 1000 è pari a 1 secondo
  (1 sec. = 1000 msec.).
- Il valore minimo è di 1 millisecondo.

```
delay(100); // ritardo di 100 msec. = 0,1 sec.
int tempo_ritardo = 250; // specifica la variabile del tempo di ritardo
delay(tempo_ritardo); // ritardo di 250 msec. = 0,25 sec.
```

Tipi di variabili utilizzate nel linguaggio C (compilatore Arduino)

variabile

Una variabile rappresenta un dato che **può cambiare il proprio valore** durante l'esecuzione del programma.

costante

Una costante rappresenta un dato che **non può cambiare di valore** nel corso dell'esecuzione.
La dichiarazione di una costante associa ad un identificatore (nome della costante) un valore (espresso eventualmente mediante altra costante).

La dichiarazione di una variabile e di una costante è un passaggio obbligatorio nel linguaggio C e richiede di definire un **identificatore** (**nome della variabile**), un **tipo** (esempio: int, char, etc) e eventualmente le dimensioni (solo per gli "array" e le "stringhe") **prima che venga utilizzata nel programma**.

Tipi di variabili utilizzate nel linguaggio C (compilatore Arduino)

boolean variabile binaria. Sono possibili solo i valori "**HIGH**" e "**LOW**" oppure "**1**" e "**0**" oppure "**TRUE**" e "**FALSE**"

char La variabile permette di memorizzare i numeri interi a **8 bit** (**1 byte**) entro un valore compreso tra **-128** e **+127**.

byte La variabile permette di memorizzare un valore numerico intero a **8 bit** (**1 byte**) senza decimali entro un valore compreso tra **0** e **255**.

int La variabile permette di memorizzare i numeri interi a **16 bit** (**2 byte**) entro un valore compreso tra **-32768** e **+32767**.

unsigned int Come la precedente ma solo valori positivi da **0** a **65535**.

long La variabile permette di memorizzare i numeri interi a **32 bit** (**4 byte**) entro un valore compreso tra **-2147483648** e **+2147483647**.

unsigned long Come la precedente ma solo valori positivi da **0** a **4294967295**.

float La variabile memorizza i numeri decimali (con virgola) in **4 byte** (32-bit) tra **$-3{,}4028235^{+38}$** e **$+3{,}4028235^{+38}$**.

Tabelle delle variabili utilizzate con Arduino

Tipo variabile	Dimensione in bit	Valori ammessi
boolean	1 bit	**TRUE** / **FALSE** oppure **HIGH** / **LOW** oppure **0** / **1**
byte	8 bit = 1 byte	solo numeri interi: **0 ÷ 255**
char	8 bit = 1 byte	solo numeri interi: **-128 ÷ 127**
unsigned char	8 bit = 1 byte	solo numeri interi: **0 ÷ 255**
int	16 bit = 2 byte	solo numeri interi: **-32768 ÷ 32767**
unsigned int	16 bit = 2 byte	solo numeri interi: **0 to 65535**
long	32 bit = 4 byte	solo numeri interi: **-2147483648 ÷ 2147483647**
unsigned long	32 bit = 4 byte	solo numeri interi: **0 ÷ 4294967295**
float	32 bit = 4 byte	numeri in virgola mobile: **-3.4028235E+38 ÷ 3.4028235E+38**

Tipi di variabili utilizzate nel linguaggio C (compilatore Arduino)

- **Esempi di variabili**

```
boolean interruttore = HIGH; // variabile intera a 1 bit (valori possibili HIGH oppure LOW)
byte numero = 10; // variabile intera a 1 byte (val. min = 0, val max. = 255)
char carattere = 0x30; // variabile intera a 1 byte (val. min = -128, val max. = 127)
int dato = -1234; // variabile intera a 2 byte (val. min = -32768, val max. = 32767)
unsigned int valore = 49034; // variabile intera a 2 byte (val. min = 0, val max. = 65536)
long i = -16000000; // variabile intera a 4 byte (val. min = -2147483648, val max. = 2147483647)
unsigned long i = 16000000; // variabile intera a 4 byte (val. min = 0, val max. = 4294967296)
float virgola = -7.23467; // variabile con virgola mobile a 4 byte (-3,4028235E+38 e +3,4028235E+38)
```

Esempi di costanti

- **Caratteri** – singolo carattere racchiuso fra apici
- **'A'** **'f'** **'6'**
- – caratteri speciali:
- **'\n'** **'\t'** **'\''** **'\\'** **'\"'**

nuova linea **tabulatore** **apostrofo** **backslash** **apici**

```
#define clock 6 // pin 6 della scheda Arduino collegato al pin 11 - SRCLK del
#define latch 7  // pin 7 della scheda Arduino collegato al pin 12 - RCLK del
#define key_74151 8 // pin 8 della scheda Arduino collegato all'output del MUX
#define CALIBRAZIONEpin 10   // digital pin 10 della scheda Arduino collegato
```

Base dei numeri in Arduino

Volendo memorizzare il numero **211** si ricorda che:

- $2\times10^2 + 1\times10^1 + 1\times10^0 = 211$ **Decimale**
- $3\times8^2 + 2\times8^1 + 3\times8^0 = 0323$ **Ottale**
- $13\times16^1 + 3\times16^0 = 0xD3$ **Esadecimale**
- $1\times2^7 + 1\times2^6 + 0\times2^5 + 1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 11010011$ **Binario**

Si avrà con **l'IDE di Arduino**:

- `int numero_decimale = 211;`
- `int numero_binario = B11010011;`
- `int numero_esadecimale = 0xD3;`
- `int numero_ottale = 0323;`

In tutte le variabili è memorizzato il valore 211

Attenzione! All'overflow della memoria RAM
(supero della massima quantità di memoria RAM disponibile)

- Anche se ogni sketch contiene le dichiarazioni delle variabili, dando di fatto la possibilità al compilatore di fare dei conteggi della quantità di memoria RAM utilizzata, con l'**IDE** di Arduino **non si ha alcun messaggio di errore se viene superata la capacità della memoria RAM disponibile (2 KByte)**, mentre se lo sketch ha dimensioni superiori alla **memoria Flash** disponibile (**32 KByte**), **questo genera un errore**.

- **Superare la RAM disponibile è un problema serio** in quanto compromette il funzionamento del programma e porta a effetti che vanno dal blocco nell'esecuzione a più subdoli bug, dovuti alla sovrascrittura delle variabili con valori errati.

Superamento della massima capacità di conteggio di una variabile di tipo «INT»


```
/* Superamento della massima capacità di
Le variabili di tipo "int" possono memori
 data: 22 luglio 2014 da G. Carpignano
void setup()
{
  Serial.begin(9600);  // inizializza la
}
void loop()
{
  int x = 32767; // definisci X come variabile di tipo "int" con valore 32767
  Serial.print("Valore iniziale memorizzato della variabile X: ");
  Serial.println(x, DEC); // stampa in decimale il numero 32767
  x++;  // corrisponde all'incremento unitario ovvero x = x + 1
  Serial.println("\nIncremeta di uno la variabile X");
  Serial.print("\nValore finale memorizzato della variabile X: ");
  Serial.println(x, DEC); // stampa in decimale il numero -32768
  x = -32768; // valore iniziale della variabile x
  Serial.print("Valore iniziale memorizzato della variabile X: ");
  Serial.println(x, DEC); // stampa in decimale il numero -32768
  x--;  // corrisponde al decremento unitario ovvero x = x - 1
  Serial.println("\nDecrementa di uno la variabile X");
  Serial.print("\nValore finale memorizzato della variabile X: ");
  Serial.println(x, DEC); // stampa in decimale il numero 32767
  while(1);  // esegui una sola volta il software
}
```

Max_count_int.ino

Le stringhe

- **Una stringa è una sequenza di caratteri delimitata da virgolette (testo in ASCII)**
- esempio: "**ciao!**" "**Hello**"
- In C le stringhe sono semplici sequenze di caratteri di cui l'ultimo, sempre presente in modo implicito, è **«\0»** (carattere di fine della stringa)
- **Esempi di testo:**

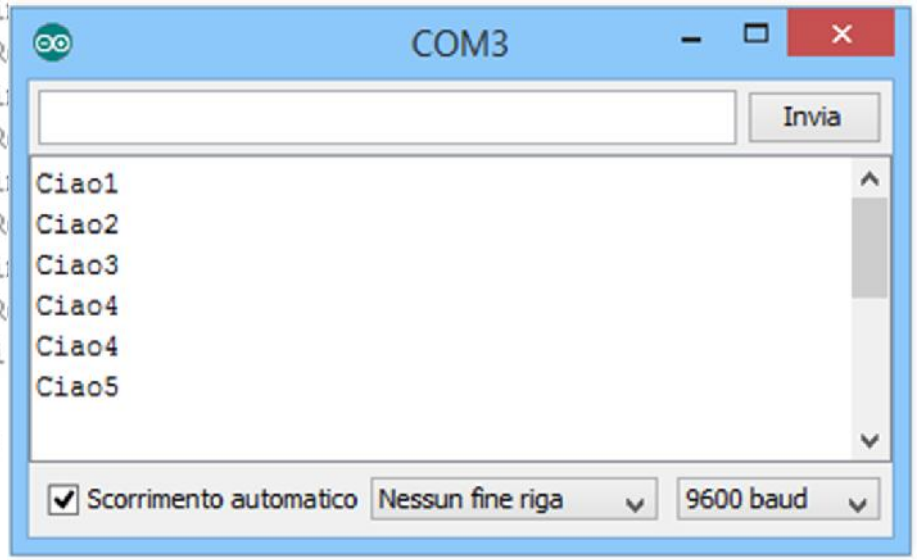
```
char testo1[6] = {'C', 'i', 'a', 'o', '1', '\0'}; // stringa con terminazione
char testo2[ ] = "Ciao2"; // stringa senza terminazione (viene chiusa in automatico dal programma)
char testo3[6] = "Ciao3"; // stringa senza terminazione (viene chiusa in automatico dal programma)
char testo4[ ] = {67, 105, 97, 111, 52, 0}; // stringa con terminazione (valori decimali)
byte testo5[ ] = { 'C', 'i', 'a', 'o', '5', '\0'}; // stringa con terminazione
```

Sketch con la stampa di testo

Stringa_1.ino

```
/*   I.I.S. Primo LEVI - Torino
  Progetto: stringhe_1.pde    Autore: G. Carpignano
  Descrizione: Stampa del testo "Ciao1", "Ciao2", etc. o stringa sulla seriale USB.
  Il carattere "\0" e' corrispondente alla fine della stringa.
  Data: 03/02/2012  */
char testo1[6] = {'C', 'i', 'a', 'o', '1', '\0'}; // stringa con terminazione
char testo2[] = "Ciao2"; // stringa senza terminazione (viene chiusa in automatico dal programma)
char testo3[6] = "Ciao3"; // stringa senza terminazione (viene chiusa in automatico dal programma)
char testo4[] = {67, 105, 97, 111, 52, 0}; // stringa con terminazione (valori decimali)
byte testo5[] = { 'C', 'i', 'a', 'o', '5', '\0'}; // stringa con terminazione
void setup() // funzione di configurazione dei Input/Output
 { // inizializza la seriale RS232 con 9600 baud
   Serial.begin(9600);
 }
void loop() // programma principale (main) --> ciclo in
 {
   Serial.write(testo1); // stampa dell'intera stringa
   Serial.println(); // stampa di un CR (carriage Return) e LF (line Feed)
   Serial.write(testo2); // stampa dell'intera stri
   Serial.println(); // stampa di un CR (carriage R
   Serial.write(testo3); // stampa dell'intera stri
   Serial.println(); // stampa di un CR (carriage R
   Serial.write(testo4); // stampa dell'intera stri
   Serial.println(); // stampa di un CR (carriage R
   Serial.print(testo4); // stampa dell'intera stri
   Serial.println(); // stampa di un CR (carriage R
   for(int x=0; x<5; x++) // ciclo con il numero di
    {
      Serial.write(testo5[x]); // stampa del singolo
    }
   while (1); // loop infinito (blocca il micro)
 }
```

Serial.print();



Frequenza di lampeggio modificata da pulsante

l'**indentazione** viene effettuata con il tasto "**TAB**" che sposta verso destra il cursore visualizzato. Rispetta l'annidamento delle varie istruzioni e aumenta la leggibilità del programma (modifica più facile).

DigitalReadSerial_2.ino

```
  Progetto: DigitalReadSerial_2      Autore: Questo è un esempio di pubblico dominio
  Descrizione: Lettura di un input digitale (pulsante collegato al pin7)
  con visualizzazione del livello logico sul led e ripetizione del ciclo all'infinito.
  Se pulsante non e' premuto il led lampeggia lentamente (1 Hz) altrimenti velocemente (10 Hz).
  Data: 28/01/2012 */
int ritardo;  // variabile utilizzata per il ritardo
void setup()   // funzione di inizializzazione della seriale RS232
{
  pinMode(7, INPUT);   // inizializza il pin 7 della scheda Arduino come INPUT (PULSANTE)
  digitalWrite(7, HIGH);  // settaggio per la resistenza interna di pull-up da 10Kohm
  pinMode(13, OUTPUT); // inizializza il pin 13 della scheda Arduino come OUTPUT (LED)
}
void loop()    // programma principale (main) --> ciclo infinito (loop)
{
  int pulsante = digitalRead(7);  // acquisisci l'input pin 7 nella variabile "pulsante"
  if (pulsante == 0) // verifica se il pulsante è premuto (condizione VERA = pulsante n.a. PREMUTO)
      ritardo = 50;  // velocità di lampeggio elevata 50+50=100 millisecondi = 1 / 0,1 = 10 Hz
  else  // altrimenti se il pulsante non è premuto (condizione FALSA = pulsante n.a. NON PREMUTO)
      ritardo = 500;  // velocità di lampeggio bassa 500+500=1000 millisecondi = 1 / 1 = 1 Hz
  digitalWrite(13, HIGH);   // accendi il LED forzando un livello ALTO sul pin 13
  delay(ritardo);  // funzione di ritardo con tempo modificato se pulsante e' premuto
  digitalWrite(13, LOW);    // spegni il LED forzando un livello BASSO sul pin 13
  delay(ritardo);  // funzione di ritardo con tempo modificato se pulsante e' premuto
}
```

Logica digitale **AND**, **OR**, **NOT**, **EX-OR** nel linguaggio C

Gli operatori logici servono per confrontare due espressioni e restituiscono un valore VERO o FALSO a seconda dell'operatore.

Tabelle di verità

AND			OR			NOT		EX-OR		
A	B	X	A	B	X	A	X	A	B	X
0	0	**0**	0	0	**0**	0	**1**	0	0	**0**
0	1	**0**	0	1	**1**	1	**0**	0	1	**1**
1	0	**0**	1	0	**1**			1	0	**1**
1	1	**1**	1	1	**1**			1	1	**0**

Ci sono 4 operatori logici "AND", "OR", "NOT" e "EX-OR" che sono spesso utilizzati nelle istruzioni "if() … else" e "while()".

Da memorizzare. Importante!! Qualsiasi numero DIVERSO da ZERO è VERO (compresi i valori negativi), quindi solo il valore ZERO è FALSO.

Logica digitale **AND**, **OR**, **NOT**, **EX-OR** nel linguaggio C

- Esempi di operazioni logiche

```
byte x=3, y=4, z=0; // Per la logica AND (&&) la condizione e' VERA solo se entrambe
if (x > 0 && x < 5) // le espressioni sono VERE. In questo esempio:
   z = 15;          //  x > 0 e' VERO perche' 3 > 0 mentre x < 5 e' VERO perche' 3 < 5.
else                // Otteniamo VERO && VERO --> VERO, cioe' tutta l'espressione
   Z = 10;          // (x > 0 && x < 5) vale VERO quindi il risultato e' che z = 15
```

```
char x=-1, y=4, z=0; // Per la logica OR (||) la condizione e' VERA quando una o
if (x > 0 || y > 8) // entrambe le espressioni sono VERE. In questo esempio:
   z = 11;          //  x > -1 e' FALSO perche' -1 > 0 mentre y > 0 e' FALSO perche' 4 > 8.
else                // Otteniamo FALSO || FALSO --> FALSO, cioe' tutta l'espressione
   z = 9;           // (x > 0 || y > 8) vale FALSO quindi il risultato e' che z = 9
```

```
char x=7, z=0;      // Per la logica NOT (!) la condizione e' VERA quando l'espressione e'
if (!x > 0)         // FALSA e viceversa. In questo esempio:
   z = 5;           // ricorda che qualsiasi numero diverso da zero e' VERO, quindi con x=7
else                // x e' VERO che diventa FALSO con la negazione !x
   z = 7;           // quindi il risultato e' che z = 7
```

Logica digitale **AND**, **OR**, **NOT**, **EX-OR** nel linguaggio C

- **Esempi di operazioni logiche sul singolo bit (bitwise)**

Sintassi	Descrizione	sensore_a		sensore_b		Risultato sensore_c
c = a | b	OR	10100101	OR	11110000	=	11110101
c = a & b	AND	10100101	AND	11110000	=	10100000
c = a ^ b	EX-OR	10100101	OR	11110000	=	11110101
c = ~a	NOT	10100101	NOT		=	01011010

Esempi

```
sensore_a = sensore_a | 0x80;
    // forza a 1 il bit 7 (msb)

if ((sensore_b & 0x81) == 0)
    // controlla se il bit 7 e il bit 0 sono a livello basso

sensore_c = sensore_c ^ 0x80;
    // commuta nel suo complemento il bit 7

sensore_d = sensore_d & (~0x80);
    // forza basso il bit 7
```

Bitwise con operatori logici

bitwise.ino

```
  Descrizione: dimostrazione del funzionamento degli operatori AND, OR, EX-OR e NOT
  in modalita' "bitwise" con stampa sulla seriale USB */
void setup() // funzione di inizializzazione della seriale USB
 { // inizializza la seriale RS232 con 9600 baud, 8 bit dati, nessuna parità e 1 bit di sto
  Serial.begin(9600);
 }
void loop() // programma principale (main) --> ciclo infinito (loop)
 {
  Serial.print("Logica AND   --> 3 & 1 uguale "); // bitwise della logica AND tra il valore
  Serial.print(3 & 1); //  stampa del risultato 1 in decimale
  Serial.print(" decimale, oppure in binario:   ");
  Serial.println(3 & 1 , BIN); // stampa in binario 00000011 & 00000001 --> 00000001
  Serial.print("Logica OR    --> 5 | 3 uguale "); // bitwise della logica OR tra il valore
  Serial.print(5 | 3 ); // stampa del risultato 7 in decimale
  Serial.print(" decimale, oppure in binario: ");
  Serial.println(5 | 3 , BIN); // stampa in binario 00000101 | 00000011 --> 00000111
  Serial.print("Logica EX-OR --> 5 ^ 3 uguale "); // bitwise della logica EX-OR tra il valo
  Serial.print(5 ^ 3); // stampa del risultato 6 in decimale
  Serial.print(" decimale, oppure in binario: ");
  Serial.println(5 ^ 3 , BIN); // stampa in binario 00000101 ^ 00000011 --> 00000110
  byte byteValore = 1;
  unsigned int intValore = 1;
  unsigned long longValore = 1;
  byteValore = ~byteValore; // bitwise della logica NOT su una variabile a 8 bit
  intValore = ~intValore; // bitwise della logica NOT su una variabile a 16 bit
  longValore = ~longValore; // bitwise della logica NOT su una variabile a 32 bit
  Serial.print("Logica NOT con variabile tipo 'BYTE = 1' --> ~byteValore (8 bit) uguale ");
  Serial.println(byteValore, BIN); // stampa il risultato in binario
  Serial.print("Logica NOT con variabile tipo 'INT = 1'  --> ~intValore (16 bit) uguale ");
  Serial.println(intValore, BIN); // stampa il risultato in binario
  Serial.print("Logica NOT con variabile tipo 'LONG = 1' --> ~intValore (32 bit) uguale ");
  Serial.println(longValore, BIN); // stampa il risultato in binario
  while(1);  // loop infinito --> blocca il programma
 }
```