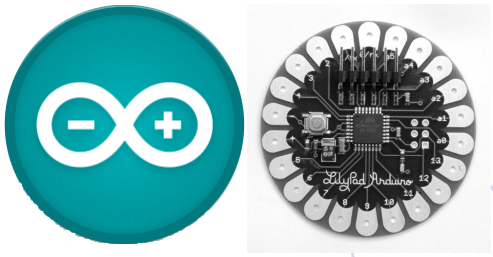


Progetti reali con ARDUINO

Introduzione alla scheda Arduino (parte 2ª) ver.1

Classe 3BN (elettronica)

marzo 2012 – Giorgio Carpignano

I.I.S. Primo LEVI - TORINO

Il menù per oggi

Lettura dei pulsanti
Comunicazione con altri dispositivi
Le istruzioni fondamentali
- if …. else
- while()
- do …. while()
- for
- switch …. case

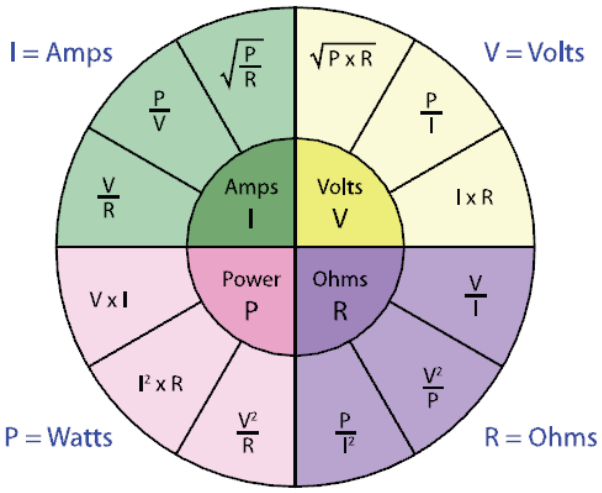
Tipi di variabili e costanti
Logica digitale AND, OR, NOT, EX-OR
Inserimento dati da tastiera del Computer
Scheda Arduino in modalità “Stand-alone”

Inoltre, tutte le domande relative alla scorsa settimana?

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:

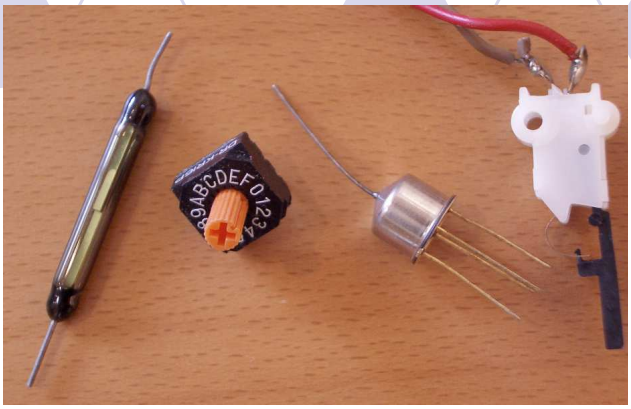
Prima di provare qualcosa di nuovo, partire da una situazione o uno stato sicuramente funzionante sia dell'Hardware che del software.

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

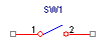
Altri tipi di interruttori e contatti.



Il **sensore di inclinazione** ha una pallina che sente il movimento.
Gli **interruttori reed** (magnetici) si chiudono in presenza di un magnetino (nella figura il 1°a sinistra).
L'interruttore esadecimale (2°a sinistra) è in realt à un deviatore con molti interruttori in uno, possiede un segnale a 4 vie.

Interruttori, Pulsanti e Deviatori

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

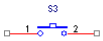


INTERRUTTORE 1 CONTATTO

INTERRUTTORE 2 CONTATTI

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

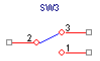
- Pulsante di tipo **normalmente aperto N.A.**
- Pulsante di tipo **normalmente chiuso N.C.**



PULSANTE NORMALMENTE APERTO

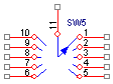
PULSANTE NORMALMENTE CHIUSO

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



DEVIATORE 1 CONTATTO 2 POSIZIONI

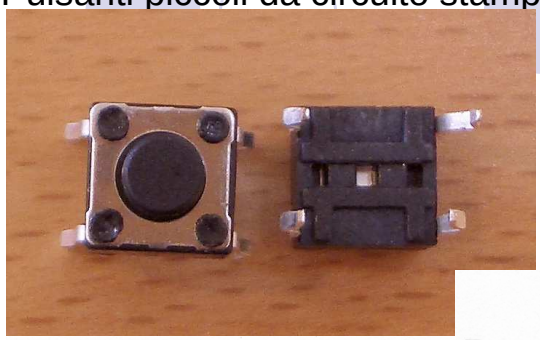
DEVIATORE 2 CONTATTI 2 POSIZIONI

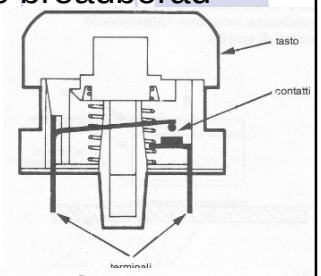


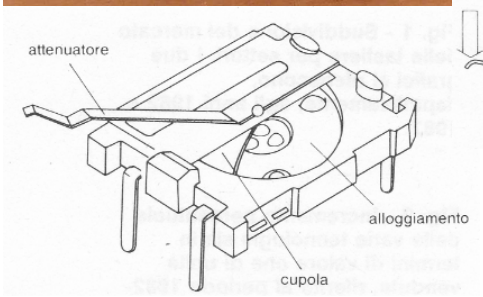
DEVIATORE 1 CONTATTO 10 POSIZIONI

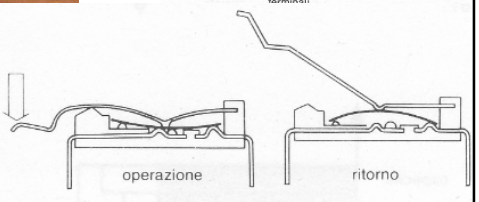
Negli schemi elettrici tutti gli organi in movimento vengono disegnati per convenzione in condizione di **riposo** (senza attivarli)

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 "vedere" una tensione:

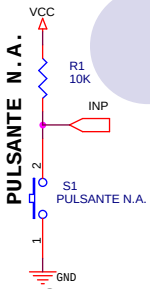
1) Un livello logico **alto** → **HIGH** → **+5V** = **VCC**

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)

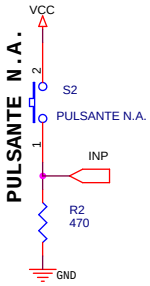


Circuito con resistenza di **pull-up** per collegare un pulsante di tipo **N.O.** (***normaly open***) a un pin del microcontrollore.

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.

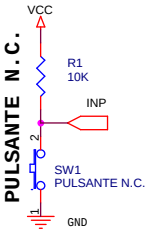
Pulsante **premuto** → livello logico in uscita **1**

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

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

Collegamento dei pulsanti N.C.

(normalmente chiusi)

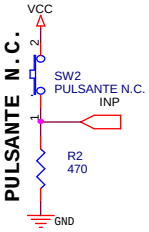


Circuito con resistenza di **pull-up** per collegare un pulsante di tipo **N.C.** (***normaly close***) a un pin del microcontrollore.

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.

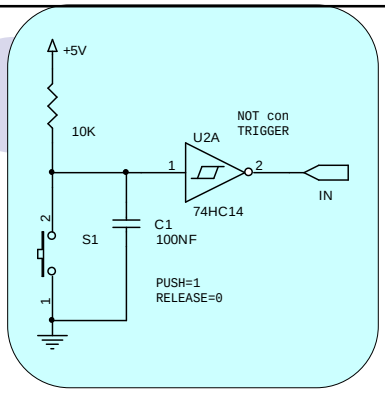
Pulsante **premuto** → livello logico in uscita **0**

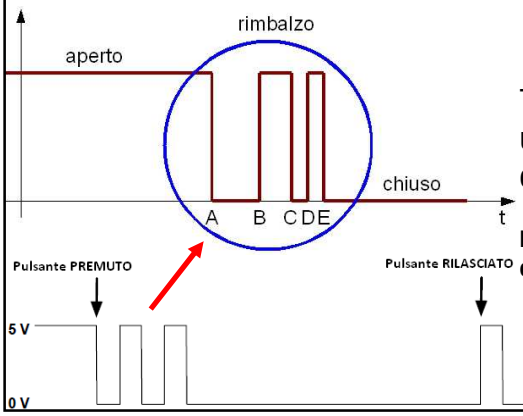
Pulsante **rilasciato** → livello logico in uscita **1**

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

Circuito elimina rimbalzi

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.





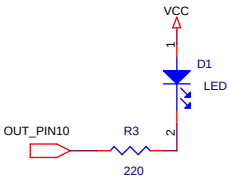
Tipici rimbalzi dei contatti di un pulsante in chiusura della durata di 1÷10ms.

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

Accensione dei led con Arduino

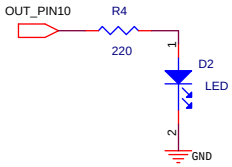
- Ogni pin è in grado di fornire circa **40 mA** di corrente, questa corrente è sufficiente per lavorare con un diodo LED (**max. 20 mA**). Valori assorbiti o erogati che sono superiori ai 40 mA o tensioni superiori a 5V su qualsiasi pin possono danneggiare il microcontrollore o il dispositivo collegato.

Led acceso con un livello **LOW**




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

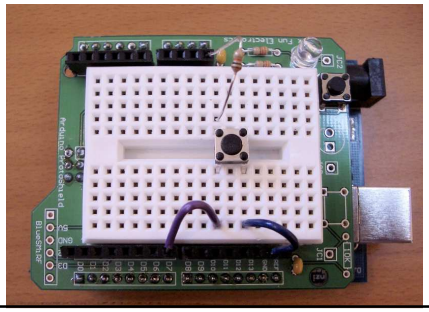
Led acceso con un livello **HIGH**

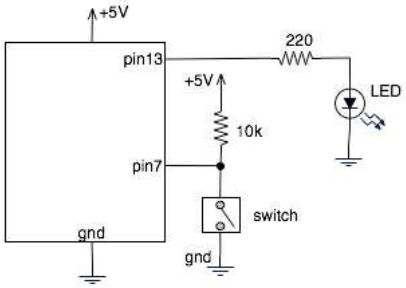



```
OUT_PIN10 = HIGH --> LED ACCESO
OUT_PIN10 = LOW --> LED SPENTO
```

Arduino con l'input/output digitale

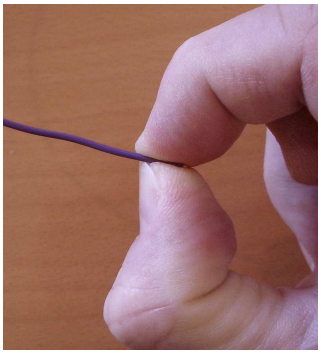
- Come **INPUT** è possibile collegare e configurare qualsiasi pulsante o interruttore tra i pin **2** e **12** della scheda [sono da escludere i pin **0** (**RX**), **1** (**TX**) e **13** (**led interno**)]
- Come **OUTPUT** è possibile collegare e configurare qualsiasi led tra i pin **2** e **13**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

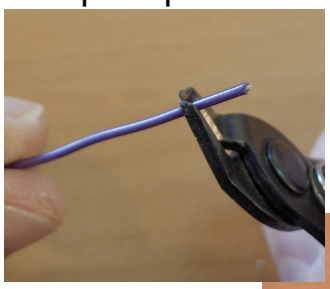


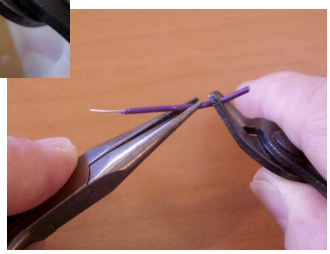


Come effettuare i collegamenti con il cavo

- 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)

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

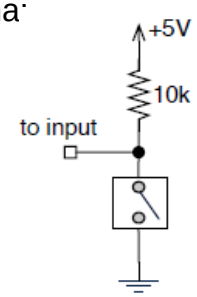
In **loop()** utilizzare **digitalRead(numero_pin);** per ottenere il livello logico acquisito sull'input (pulsante, interruttore, ecc.)

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

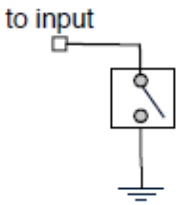
es.: **leggi_pulsante = digitalRead(7);** // leggi il valore dall'input collegato al pin7 (i valori sono "0" oppure "1")

Pulsanti e interruttori senza Resistori di pull-up esterni

Invece di utilizzare questo schema:



Potete lavorare con questo:



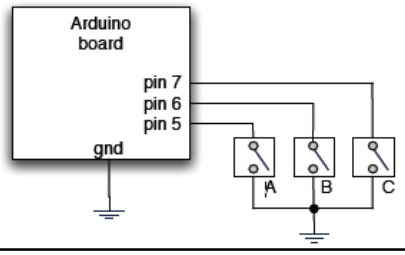
- ma come si effettua la programmazione delle resistenze interne di pull-up?

Pulsanti e interruttori senza Resistori di pull-up esterni

Risposta. utilizzando l'istruzione **digitalWrite(numero_pin, HIGH);** all'interno della funzione **setup()**

```
int puls_1 = 5; //  pulsante n.a. collegato al pin 5
int puls_2 = 6; //  pulsante n.a. collegato al pin 6
int puls_3 = 7; //  pulsante n.a. collegato al pin 7
void setup()   // funzione di inizializzazione della seriale RS232
{
 pinMode(puls_1, INPUT); // inizializza il pin 5 come INPUT collegato al pulsante n.a.
 pinMode(puls_2, INPUT); // inizializza il pin 6 come INPUT collegato al pulsante n.a.
 pinMode(puls_3, INPUT); // inizializza il pin 7 come INPUT collegato al pulsante n.a.
 digitalWrite(puls_1, HIGH); // abilita la R=10Kohm interna di pull-up sull'input pin 5
 digitalWrite(puls_2, HIGH); // abilita la R=10Kohm interna di pull-up sull'input pin 6
 digitalWrite(puls_3, HIGH); // abilita la R=10Kohm interna di pull-up sull'input pin 7
}
```

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



Effettua un Break di 10 secondi

Fine del Break!!

Comunicare con gli altri

- Arduino può utilizzare lo stesso cavo USB per la programmazione e per parlare con i computer
- Parlare ad altri dispositivi utilizzando i comandi della "Seriale"
- • **Serial.begin()** – predispone i parametri della seriale
- • **Serial.print()** – per inviare i dati al computer
- • **Serial.read()** - per leggere i dati inviati dal computer

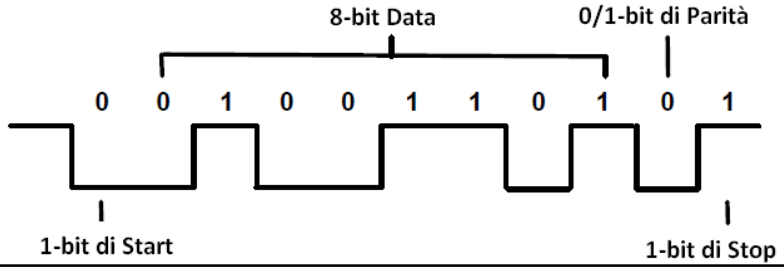
Utilizzo della funzione **Serial.print()**

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.



Invio dati al Computer

Esempi di **Serial.print()** e **Serial.println()**

- **int valore = 33**; // valore numerico della tabella ASCII
- **Serial.print(valore, BYTE);** // stampa il carattere “**!**”
- **Serial.print(valore);** // stampa i caratteri “**33**”.
  // Di default è il DECIMALE
- **Serial.print(valore, DEC);** // stampa i caratteri “**33**”.
- **Serial.print(valore, HEX);** // stampa i caratteri “**21**”.
  // Valore in esadecimale (base 16)
- **Serial.print(valore, OCT);** // stampa i caratteri “**41**”.
  // Valore in ottale (base 8);
- **Serial.print(valore, BIN);** // stampa i caratteri
  // “**100001**”. Valore in binario (base 2)

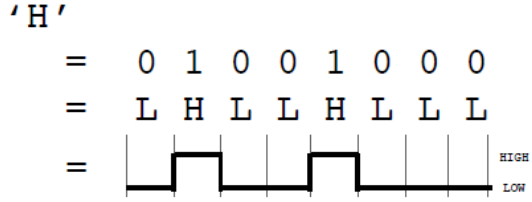
- Stesse modalità con la funzione “**Serial.println()**” con il cursore che salta su una nuova riga a capo.

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

- TX - invio dati al PC
- RX – ricezione dati dal PC
- Usato durante la programmazione per la comunicazione

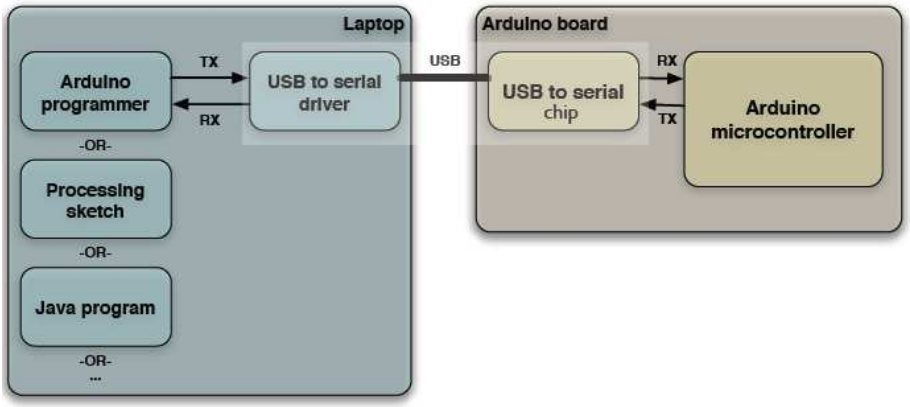
Comunicazione seriale

- "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 per permettere alla corrente del generatore di confluire allo stesso. In rari casi si utilizza un solo cavo perché il ritorno viene effettuato con un collegamento denominato "a terra".



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 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.

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.
- E' 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
```

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

Sketch con input digitale

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 pubblico dominio
  Descrizione: Lettura di un input digitale (pulsante collegato al pin7)
  con stampa del livello logico sulla porta seriale e ripetizione del ciclo all'infinito. */
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
   }
}
```

Utilizzo della funzione **delay()**

- Mette in pausa un programma per la quantità di tempo specificato in millisecondi, 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).

Tipi di variabili utilizzate nel linguaggio C (compilatore Arduino)

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

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**.

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 4711 si ricorda che:

- $4\times10^3 + 7\times10^2 + 1\times10^1 + 1\times10^0 = 4711$
- $1\times8^4 + 1\times8^3 + 1\times8^2 + 4\times8^1 + 7\times8^0 = 011147$
- $1\times16^3 + 2\times16^2 + 6\times16^1 + 7\times16^0 = 0x1267$

Si avrà con **l'IDE di Arduino**:

- int numero_decimale = 4711;
- int numero_binario = B1001001100111;
- int numero_ottale = 011147;
- int numero_esadecimale = 0x1267;

Le stringhe

- **Una stringa è una sequenza di caratteri delimitata da virgolette**
- esempio: "**ciao**" "**Hello\n**"
- 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)
- **La stringa "ciao" verrà inserita come:**

byte stringa_1[ ] = { 'c', 'i', 'a', 'o', '\0'};

Sketch con input digitale modificato

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

```
/*  I.I.S. Primo LEVI - Torino
  Esercizio N. 3 bis
  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 sulla 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 il valore dell'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	Variabile **a**		Variabile **b**		Risultato Variabile **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

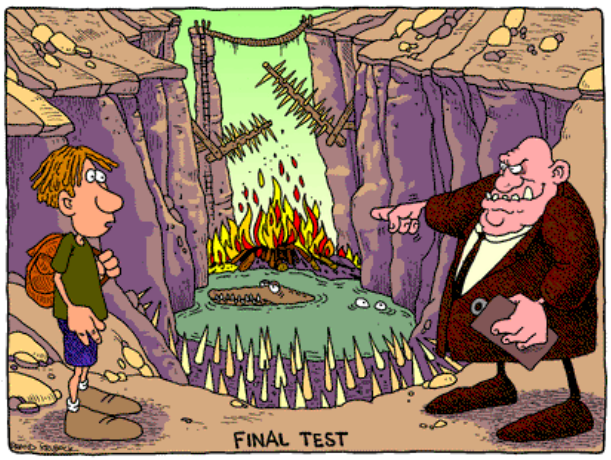
sensore_a = sensore_a | 0x80; // forza a 1 il bit 7 (msb)

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

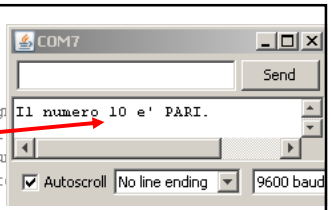
sensore_d = sensore_d & (~0x80); // forza basso il bit 7

Esercizio da svolgere subito!



Scrivere un programma in modo tale che venga inserito un numero intero nella variabile denominata "valore" e stabilisca se il numero e' **pari** o **dispari**.

Esercizio pari o dispari (1° metodo)




```
/*  I.I.S. Primo LEVI - Torino     Data: 03/12/2010
  Esercizio N. 5     Progetto: pari_dispari_1     Autore: G. Carpig
  Descrizione: Scrivere un programma in modo tale che venga inseri
  intero nella variabile denominata "valore" e stabilisca se il nu
byte valore = 10; // variabile a 8 bit con il numero da controllar
void setup()
{// inizializza la seriale RS232 con 9600 baud, 8 bit dati, nessuna parità e 1 bit di stop
  Serial.begin(9600);
}

void loop() // programma principale
{ // la condizione viene verificata se e' uguale a 0, ma prima viene effettuata l'AND (&)
  // con la variabile "valore", cioe' si avra' 10 & 1 che vale in binario "00001010" & "00000001"
  // diventa, effettuando la logica AND su ogni singolo bit "00000000" che corrisponde a 0.
  if ((valore & 1) == 0)  // se il confronto vale 0
    {  // si e' in presenza di un numero PARI
      Serial.print("Il numero ");
      Serial.print(valore, DEC); // trasmissione sulla seriale del numero
      Serial.print(" e' PARI.");
     }
  else
    {  // altrimenti il numero e' DISPARI
      Serial.print("Il numero ");
      Serial.print(valore, DEC); // trasmissione sulla seriale del numero
      Serial.print(" e' DISPARI.");
     }
  while(1);  // loop infinito (blocca il micro con questa istruzione)
}
```

Esercizio pari o dispari (2° metodo)

```
/*  I.I.S. Primo LEVI - Torino     Data: 03/12/2010
  Esercizio N. 5     Progetto: pari_dispari_2     Autore: G. Carpignano
  Descrizione: Scrivere un programma in modo tale che venga inserito un numero
  intero nella variabile denominata "valore" e stab                              .*/
byte valore = 10; // variabile con il numero da con
void setup()
{// inizializza la seriale RS232 con 9600 baud, 8 b                    bit di stop
  Serial.begin(9600);
}

void loop() // programma principale
{
  if ((valore % 2) == 0)  // se il resto della divisione per 2 vale 0
    {  // si e' in presenza di un numero PARI
      Serial.print("Il numero ");
      Serial.print(valore, DEC); // trasmissione sulla seriale del numero
      Serial.print(" e' PARI.");
     }
  else
    {  // altrimenti il numero e' DISPARI
      Serial.print("Il numero ");
      Serial.print(valore, DEC); // trasmissione sulla seriale del numero
      Serial.print(" e' DISPARI.");
     }
  while(1);  // loop infinito (blocca il micro con questa istruzione)
}
```

Il carattere " %" permette di calcolare il modulo, ovvero il resto della divisione.
In questo esempio il resto della divisione per 2 può valere solo "0" oppure "1".

Istruzione while()

- L'espressione presente all'interno della parentesi tonda (**condizione di ripetizione**) viene valutata all'inizio di ogni ciclo.
- Se la condizione risulta **VERA** si eseguono tutte le istruzioni presenti tra le parentesi graffe.
- Se la condizione risulta **FALSA** (cioè se è uguale a zero) il programma salta all'esecuzione della prima istruzione dopo la parentesi graffa chiusa.
- Se inizialmente la condizione ha valore zero, il corpo del ciclo non viene mai eseguito.
- In generale, *non è noto* ***quante volte*** l'istruzione sarà ripetuta.
- (Attenzione che qualsiasi valore memorizzato in una variabile purchè sia diverso da zero è VERO).

Istruzione **while()** con esempio

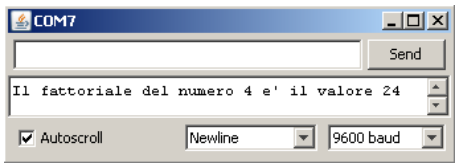
```
/*  I.I.S. Primo LEVI - Torino    Data: 03/12/2010
  Esercizio N. 7    Progetto: while_1     Autore: G. Carpignano
  Descrizione: Controllare se un input digitale (pulsante collegato al pin 7)
   e' premuto, e per tutto il tempo che rimane tale accendere un led
   (collegato al pin 13), mentre se il pulsante viene rilasciato spegnere il led.*/
int led = 13; // definizione della variabile "led" utilizzata per scrivere sul pin 13
int pulsante = 7; // definizione della variabile "pulsante" utilizzata per leggere sul pin 7
void setup() // funzione di inizializzazione dei INPUT/OUTPUT
{
  pinMode(led, OUTPUT); // inizializza il pin 13 come OUTPUT collegato al led
  pinMode(pulsante, INPUT); // inizializza il pin 7 come INPUT collegato al pulsante n.a.
  digitalWrite(pulsante, HIGH); // utilizza la R=10K di pull-up interna al microcontrollore
}
void loop()  // programma principale (main) --> ciclo infinito (loop)
{
  while(digitalRead(pulsante) == 1) // acquisisci il valore del pulsante pin 7 se il pulsante
    {  // NON E' PREMUTO si avra' un livello ALTO quindi si deve spegnere il led
     digitalWrite(led, LOW); // spegni il LED collegato al pin 13 della scheda Arduino
    }
  digitalWrite(led, HIGH); // accendi il LED collegato al pin 13 della scheda Arduino
}
```

Istruzione **do ... while()**

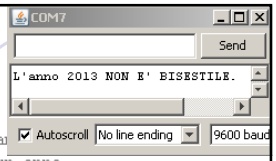
- La condizione di ripetizione viene verificata
  - **alla fine di ogni ciclo**
- Le istruzioni presenti tra le parentesi graffe **vengono sempre eseguite almeno una volta.**

Istruzione **do … while()** con esempio

```
/*  I.I.S. Primo LEVI - Torino
  Esercizio N. 9            Data: 03/02/2012
  Progetto: fattoriale_do_while       Autore: G. carpignano
  Descrizione: calcola e stampa il fattoriale con l'istruzione do .... while  */
void setup()   // funzione di inizializzazione della seriale RS232
{ // inizializza la seriale RS232 con 9600 baud, 8 bit dati, nessuna parità e 1 bit di stop
  Serial.begin(9600);
}
void loop()    // programma principale (main) --> ciclo infinito (loop)
{
  int fattoriale = 1; /* inizializzazione del fattoriale*/
  int numero = 4; // valore massimo del fattoriale da calcolare
  int i=0; /* inizializzazione del contatore*/
  do
  {  // calcolo del numero fattoriale (ad esempio per il num. 4 si avra' 1*2*3*4 = 24
    fattoriale = (i + 1) * fattoriale;
    i = i + 1;
  } while (i < numero);
  Serial.print("Il fattoriale del numero ");
  Serial.print(numero, DEC);
  Serial.print(" e' il valore ");
  Serial.print(fattoriale, DEC);
  while (1);  // blocca il programma (loop infinito)
}
```



Esempio

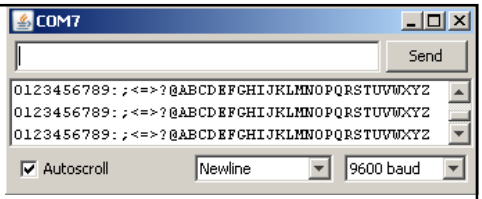



```
/*  I.I.S. Primo LEVI - Torino    Data: 03/12/2010
  Esercizio N. 8    Progetto: anno_bisestile_1     Autore: G. Carpignar
  Descrizione: Scrivere un programma in modo tale che venga inserito un anno
  nella variabile denominata "anno" e stabilisca se e' bisestile.*/
int anno = 2013; // variabile a 16 bit con l'anno da controllare se bisestile
void setup()
{// inizializza la seriale RS232 con 9600 baud, 8 bit dati, nessuna parità e 1 bit di stop
  Serial.begin(9600);
}
void loop() // programma principale
{ // la condizione viene verificata se e' uguale a 0, ma prima viene effettuata l'AND (&)
  // con la variabile "anno", cioe' si avra' 2012 & 3 che vale in binario
  // "11111011101" & "00000000011" diventa, effettuando la logica AND su ogni singolo bit
  // "00000000001" che corrisponde a 1. Si ricorda che solo gli anni divisibili per 4, cioe'
  // solo quegli anni che divisi per 4 danno come resto 0 sono anni bisestili.
  if ((anno & 0x03) == 0)  // se il confronto vale 0
    {  // si e' in presenza di un ANNO BISESTILE
      Serial.print("Il numero "); Serial.print(anno, DEC); // trasmissione sulla seriale dell'anno
      Serial.print(" E' BISESTILE.");
     }
  else
    {  // altrimenti di un ANNO NON BISESTILE
      Serial.print("L'anno "); Serial.print(anno, DEC); // trasmissione sulla seriale dell'anno
      Serial.print(" NON E' BISESTILE.");
     }
  while(1);  // loop infinito (blocca il micro con questa istruzione)
}
```

Istruzione **for**

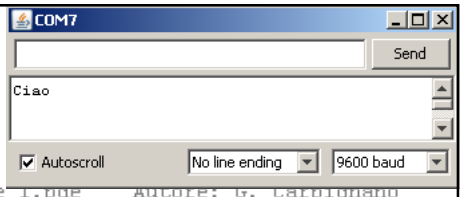
- È una istruzione di ripetizione particolarmente adatta per realizzare un numero predefinito di *cicli tramite un contatore*.
- La prima espressione è di inizializzazione (**x=0;**) viene eseguita una volta sola, prima di entrare nel ciclo.
- La seconda espressione (**x<10;**) rappresenta la condizione di permanenza nel ciclo (viene valutata all'inizio di ogni iterazione).
- La terza espressione (**x++**) rappresenta l'incremento o il decremento (**x--**) per il passaggio al ciclo successivo (valutata alla fine

Istruzione **for** con esempio




```
/*  I.I.S. Primo LEVI - Torino  Esercizio N. 4    Progetto: Alfabeto   Autore: G. Carpignano
  Descrizione: effettuare la trasmissione dei caratteri ASCII compresi tra lo "0" (zero)
               e il carattere "Z" sull'interfaccia seriale RS232. */
char carattere; // variabile per memorizzare un carattere ASCII
long i; // variabile per memorizzare loop di ritardo di circa 2 secondi
void setup()   // funzione di inizializzazione della seriale RS232
{ // inizializza la seriale RS232 con 9600 baud, 8 bit dati, nessuna parità e 1 bit di stop
  Serial.begin(9600);
}
void loop()    // programma principale (main) --> ciclo infinito (loop)
{
  // ciclo con inizio dal valore 30 Hex (coincidente con il carattere "0" zero)
  // al valore 5A Hex = 5B Hex - 1 (coincidente con il carattere "Z")
  for (carattere = 0x30; carattere < 0x5B; carattere++)
  {
    Serial.print(carattere, BYTE); // trasmissione sulla seriale del carattere in codice ASCII
  }
  delay(1000);  // ciclo di ritardo di 1 secondo
  Serial.print("\n"); // trasmissione sulla seriale del carattere "new line" (nuova linea)
}
```

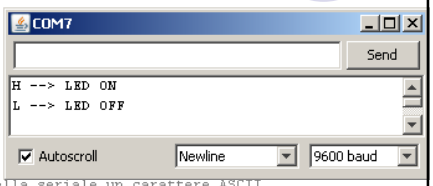
Esempio con le stringhe




```
/*   I.I.S. Primo LEVI - Torino
  Esercizio N. 11   Progetto: stringhe_1.pde    Autore: G. Carpignano
  Descrizione: Stampa la stringa "Ciao" sulla seriale.
  Data: 03/02/2012  */
  byte stringa_1[] = { 'C', 'i', 'a', 'o', '\0'};
void setup() // funzione di configurazione dei Input/Output
{ // inizializza la seriale RS232 con 9600 baud
  Serial.begin(9600);
}
void loop() // programma principale (main) --> ciclo infinito (loop)
{
  for(int x=0; x<5; x++) // ciclo con il numero di caratteri da stampare
   {
    Serial.print(stringa_1[x], BYTE); // stampa della stringa
   }
  Serial.println(" ");
  while (1); // loop infinito (blocca il micro)
}
```

Inserimento dati da tastiera del Computer alla scheda Arduino. Come fare?

Nel software se si digita "**H**" il led si **accende**, mentre se si digita "**L**" si **spegne**. Qualsiasi altro carattere viene ignorato.

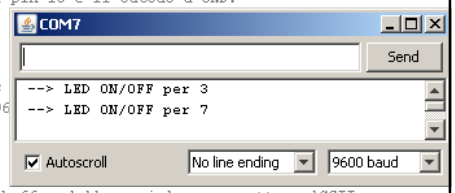



```
int led = 13; // il led e' collegato con l'Anodo sul pin 13 e il Catodo a GND.
int leggi_byte;
void setup()
{
  pinMode(led, OUTPUT);  // configura il pin 13 come output
  Serial.begin(9600);  // inizializza la seriale a 9600 baud
}
void loop()
{
  if (Serial.available() > 0) // se e' presente sul buffer della seriale un carattere ASCII
  {
    leggi_byte = Serial.read(); // acquisisci il carattere dalla seriale e memorizzalo
    // se il byte letto dalla seriale e' coincidente con il carattere maiuscolo "H" (0x48)
    // oppure con "1" (0x31) accendi il led
    if ((leggi_byte == 'H') || (leggi_byte == '1'))
      {
        digitalWrite(led, HIGH); // accendi il led
        Serial.println(leggi_byte, BYTE); // ritrasmetti il carattere sulla seriale
      }
    // se il byte letto dalla seriale e' coincidente con il carattere maiuscolo "L" (0x4C)
    // oppure con "0" (0x30) spegni il led
    if ((leggi_byte == 'L') || (leggi_byte == '0'))
      {
        digitalWrite(led, LOW); // spegni il led
        Serial.println(leggi_byte, BYTE); // ritrasmetti il carattere sulla seriale
      }
  }
}
```

Inserimento dati da tastiera del Computer

Digitare un numero da **1** a **9** e il LED deve lampeggiare per il numero di volte digitato sulla tastiera del Personal Computer

```
int led = 13; // il led e' collegato con l'Anodo sul pin 13 e il Catodo a GND.
int leggi_byte;
void setup()
{
  pinMode(led, OUTPUT);  // configura il pin 13 come
  Serial.begin(9600);  // inizializza la seriale a 96
}
void loop()
{
  if (Serial.available() > 0) // se e' presente sul buffer della seriale un carattere ASCII
  {
    leggi_byte = Serial.read(); // acquisisci il carattere dalla seriale e memorizzalo
    if ((leggi_byte > '0') && (leggi_byte <= '9')) // se il valore letto e' > 0 e <= 9
      {
        leggi_byte = leggi_byte - '0'; // converti valore da ASCII a valore numerico
        Serial.print(" --> LED ON/OFF per ");
        Serial.println(leggi_byte, DEC); // ritrasmetti il numero sulla seriale
        // ripeti la sequenza di accensione/spegnimento per il numero inserito da tastiera del PC
        for (int i = 0; i < leggi_byte; i++)
            {
              digitalWrite(led, HIGH); // accendi il led
              delay (100); // ritardo di 0,1 secondi
              digitalWrite(led, LOW); // spegni il led
              delay (100); // ritardo di 0,1 secondi
            }
      }
  }
}
```



Istruzione **switch ….. case**

Consente di selezionare l'esecuzione tra gli N blocchi di istruzioni componenti, in base al valore di una espressione (solo con variabili intere, cioè senza virgola).

Per terminare ogni "**case**" si utilizza l'istruzione "**break**" (che provoca l'uscita forzata dallo switch).

È possibile specificare un'etichetta "**default**". Essa viene eseguita per qualunque valore diverso dai valori specificati in precedenza nei vari "case".

Istruzione **switch ….. case** con esempio

Digitare un numero da **1** a **4** e il LED deve lampeggiare per il numero di volte digitato sulla tastiera del Personal Computer

```
int led = 13; // il led e' collegato con l'Anodo sul pin 13 e il Catodo a GND.
int leggi_byte, num_flash;
void setup()
{
  pinMode(led, OUTPUT);  // configura il pin 13 come output
  Serial.begin(9600);  // inizializza la seriale a 9600 baud
}
void loop()
{
  while (Serial.available()); // controlla se e' presente nel buffer della seriale un
  leggi_byte = Serial.read(); // acquisisci il carattere dalla seriale e memorizzalo
  switch (leggi_byte)  // confronta con i possibili valori
  {
    case '1': // caso relativo alla ricezione del carattere ASCII 1
     num_flash = 1; // numero di volte che deve lampeggiare
    break;
    case '2': // caso relativo alla ricezione del carattere ASCII 2
      num_flash = 2; // numero di volte che
    break;
    case '3': // caso relativo alla ricezion
      num_flash = 3; // numero di volte che
    break;
    case '4': // caso relativo alla ricezion
      num_flash = 4; // numero di volte che
    break;
    default: // caso relativo alla ricezione
    break;
  }  // fine switch
}
Serial.print(" --> LED ON/OFF per ");
Serial.println(num_flash, DEC); // ritrasmetti il numero
// ripeti la sequenza di accensione/spegnimento per il n
for (int i = 0; i < num_flash; i++)
   {
      digitalWrite(led, HIGH); // accendi il led
      delay (100); // ritardo di 0,1 secondi
      digitalWrite(led, LOW); // spegni il led
      delay (100); // ritardo di 0,1 secondi
   }
```

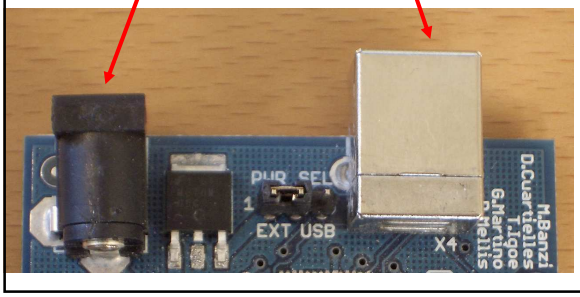
Scheda Arduino in modalità “Stand-alone”

- ***Stand-alone*** è un termine inglese che può essere tradotto letteralmente come "**a sé stante**", e significa quindi "**indipendente**".
- In informatica, l'espressione **stand-alone** indica che un oggetto o un software è capace di funzionare da solo o in maniera indipendente da altri oggetti o software, con cui potrebbe altrimenti interagire.
- È ovvio che la completa indipendenza si ottiene solo con una alimentazione esterna di tipo trasportabile.

Alimentazione della scheda Arduino

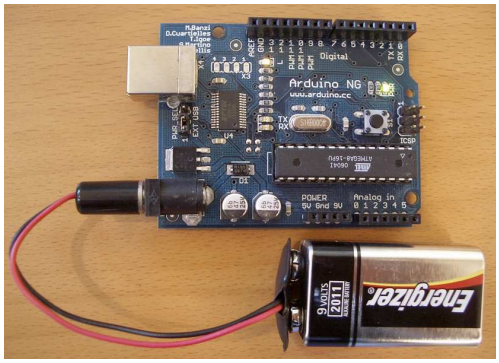
Arduino può essere alimentato tramite:

- Interfaccia USB (+5V)
- Alimentatore esterno (+9÷15V con contatto centrale collegato al positivo e corrente > 300 mA)



Alimentazione esterna da batteria

- Un metodo veloce e semplice per alimentare la scheda Arduino
- L'ingresso è protetto contro la polarità invertita da un diodo



Durata della batteria?

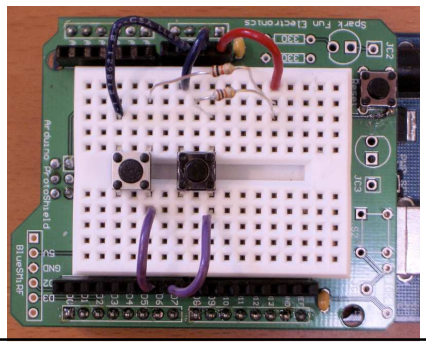
- La sola scheda Arduino richiede una corrente di circa 40 mA
- Ogni led aggiunto assorbe circa 20 mA quando viene acceso
- Ogni servo motore richiede una corrente media di circa 100÷150 mA
- le resistenze di pull-up dei pulsanti, interruttori e dei potenziometri assorbono quasi 0 mA
- La batteria da 9V possiede una capacità media espressa in milliampere all'ora (mA/h) di **400 mA/h**
- Quindi alimentando la sola scheda Arduino si avrà:

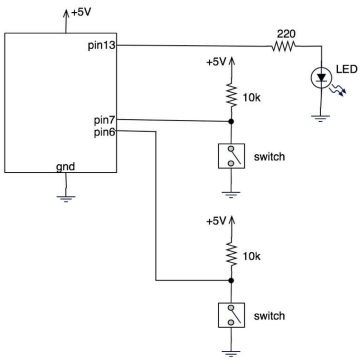
  400 mA/h / 40 mA = 10 ore di ininterrotto funzionamento.
- Ovviamente dovendo alimentare altri circuito il tempo si riduce ulteriormente in funzione del loro assorbimento medio richiesto.
- Nel caso si richieda un tempo maggiore di corretto funzionamento si ha a disposizione due tecniche di funzionamento:
  - 1) disporre il microcontrollore in modalità "sleep" (max assorbimento di pochi μA.
  - 2) disporre di una batteria di capacità superiore magari collegando due batterie in parallelo.

Doppi Pulsanti

Prova a risolvere il seguente problema:

- Il led in condizione iniziale risulta spento.
- Il led si accende per tutto il tempo che entrambi i pulsanti sono premuti.
- Se viene premuto uno solo dei due pulsanti il led rimarrà spento.





Prossima settimana

- Movimento tramite i “Servo”
- Utilizzo di un LED RGB per ottenere i colori della luce
- Controllo della scheda Arduino da un computer
- Controllo di un computer con la scheda Arduino