Arduino

L'utilizzo di un microcontrollore presenta alcune difficoltà a chi vi si avvicina per la prima volta: la necessità di realizzare una basetta stampata su cui montare il microcontrollore e i componenti accessori, la conoscenza del linguaggio assembler e le relative tecniche di programmazione, ecc.
I costruttori, per agevolare il lavoro ai progettisti, in genere mettono a disposizione sistemi di sviluppo costituiti da piattaforme hardware e ambienti software per la programmazione.

Nel 2005 un gruppo di progettisti dell'Interaction Design Institute di Ivrea, ha messo a punto un sistema di sviluppo, il cui nome Arduino è ispirato da quello del bar che erano soliti frequentare, con l'obiettivo della massima economicità e massima semplicità di utilizzo, in modo da poter essere impiegato anche da persone con limitate competenze in campo elettronico-informatico.
Questo sistema si è poi diffuso in tutto il mondo, soprattutto in ambito didattico e hobbistico.
Sui siti **online.scuola.zanichelli.it/mirandola** e **online.scuola.zanichelli.it/mirandolacorso**, nella sezione PROGETTI CON ARDUINO che sarà regolarmente aggiornata, si forniscono indicazioni per progetti svolti, o da sviluppare, mediante la scheda Arduino.

Arduino mette a disposizione:

- una piattaforma hardware: è costituita da schede basate su microcontrollori Atmel, che integrano anche vari dispositivi che ne semplificano l'utilizzo e il collegamento all'esterno, come il quarzo per la generazione del clock, il regolatore di tensione, l'interfaccia per la comunicazione USB, connettori di input e output per segnali analogici e digitali, LED, ecc.; nella figura 1 è rappresentata la scheda Arduino Uno a cui ci si riferisce nella presente guida.
- Un ambiente di sviluppo integrato (IDE: *Integrated Development Environment*) in cui si scrive il software con un linguaggio, detto *Wiring*, derivato dal C/C++, che mette a disposizione gli strumenti per scrivere il programma (detto *sketch*) e controllarne la sintassi (*editor*), per convertire il programma dal linguaggio sorgente al codice macchina (*compilatore*), per caricare il codice nella memoria del microcontrollore (*loader*), per controllare l'esecuzione del software e correggerne eventuali errori (*debugging*), ecc.

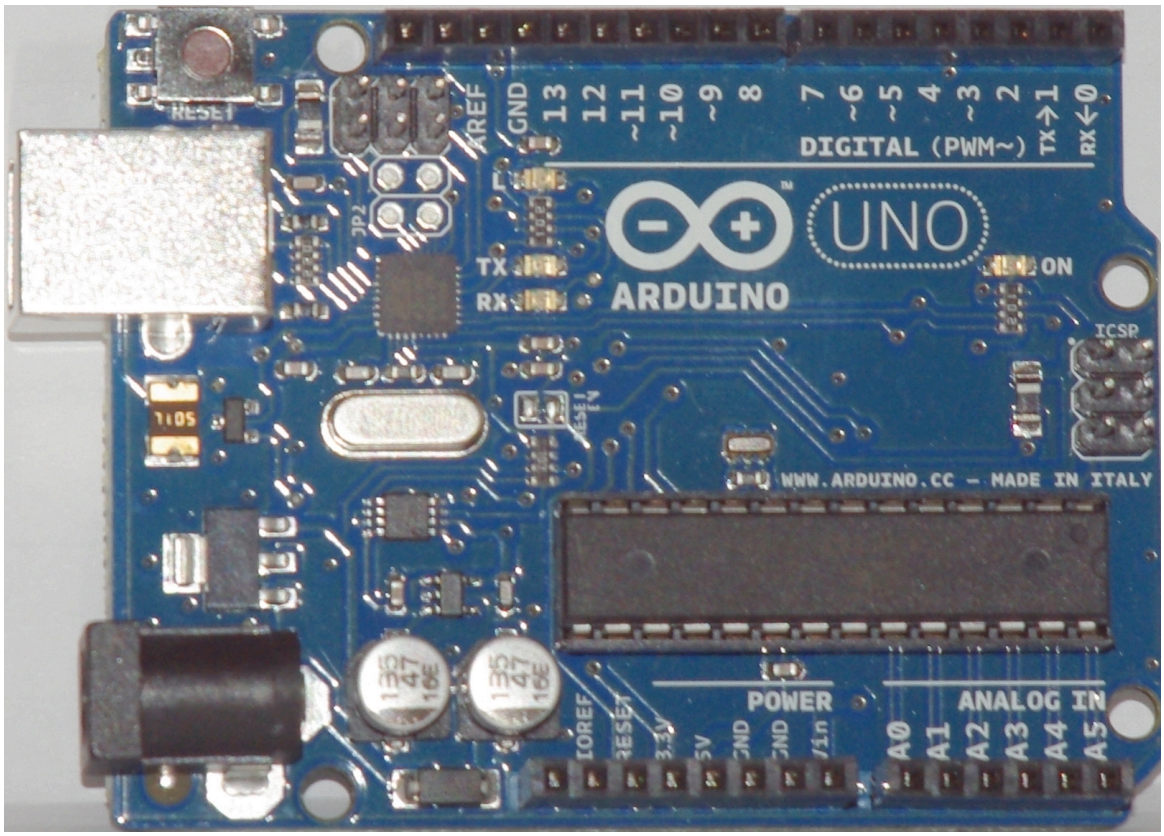

figura 1 - La scheda Arduino Uno, con il µC Atmel ATmega328, nel contenitore DIP.

I pregi principali di questo sistema sono:

- l'economicità dell'hardware e la gratuità del software, scaricabile dal sito web di Arduino *http://arduino.cc*;
- la scheda contiene già tutto ciò che serve per un utilizzo di base;
- la semplicità di collegamento alla scheda, sia verso il computer (USB) sia verso gli altri circuiti elettronici (mediante pin messi a disposizione sulla scheda);
- la relativa semplicità della programmazione in C, che non richiede quindi la conoscenza del linguaggio assembler del microcontrollore;
- la grande disponibilità sul web di programmi (detti *sketch*), per svolgere le più svariate funzioni, o di progetti interamente sviluppati;
- la disponibilità di varie librerie di funzioni che semplificano il pilotaggio di display LCD, motori, trasduttori, servomotori, ecc.;
- la disponibilità di schede aggiuntive (dette *shield*), realizzate da vari produttori, per interfacciare la scheda Arduino a motori, a carichi di potenza, a trasmettitori wireless, alla rete internet, ecc.

Dal punto di vista funzionale la scheda Arduino presenta i collegamenti rappresentati in figura 2 e individuabili in figura 1:

- la presa USB per il collegamento al computer, durante la programmazione e la messa a punto del software;
- la presa per l'alimentazione (7 -12 V), per il funzionamento autonomo (*stand-alone*) una volta che la scheda viene scollegata dal computer e inserita nel circuito a cui è destinata; un regolatore di tensione all'interno produce la 5 V stabilizzata necessaria ai vari integrati;
- ingressi digitali (a livelli TTL) individuati, mediante opportuna programmazione, tra i 14 pin *digital*;

- ingressi analogici (*analog in*): le tensioni presenti su questi sei ingressi sono acquisite e convertite in digitale (a 10 bit) mediante opportune istruzioni;
- uscite digitali individuate, mediante opportuna programmazione, tra i 14 pin *digital*;
- uscite PWM (*Pulse Width Modulation*): alcuni dei pin digital (3, 5, 6, 9, 10, 11), se definiti come uscite, possono produrre segnali PWM mediante opportune istruzioni, per il controllo di alcuni attuatori come motori, lampade, riscaldatori, servomotori, ecc. La modulazione a larghezza d'impulso (PWM) modifica il duty-cycle di un'onda rettangolare (dallo 0% al 100%) in base al valore della grandezza modulante (in questo caso un numero compreso tra 0 e 255). L'informazione contenuta in un segnale PWM è di tipo analogico, per questo l'istruzione che imposta il segnale su uno dei pin dedicati è "*analogWrite*", nonostante tali pin siano tra quelli definiti "*digital*".

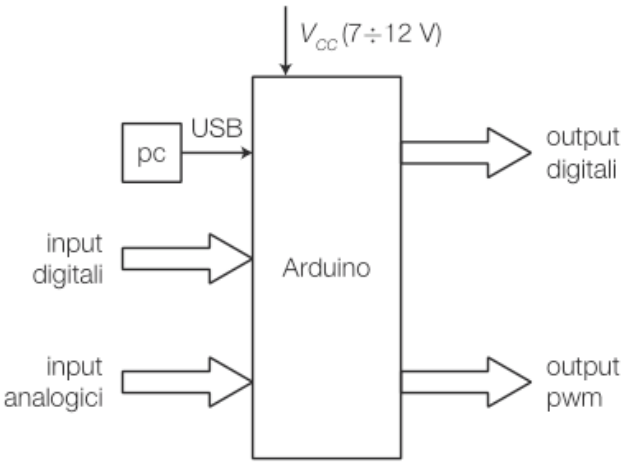

figura 2 - Principali collegamenti della scheda Arduino.

Per iniziare

1. Scaricare l'ultima versione del software Arduino dal sito *http://arduino.cc*
2. Installare il software e i driver.
3. Collegare la scheda Arduino ad una presa USB per alimentarla e programmarla; si accende il LED verde ON (una volta programmata potrà funzionare con un'alimentazione da 7 V a 12 V posta sull'apposito connettore (il + è il contatto interno).
4. Lanciare il software Arduino.
5. Aprire lo sketch di esempio che fa lampeggiare il LED *L* sulla scheda: *File* > *Examples* > 1.*Basics* > *Blink* (oppure cliccare sul pulsante ↑).
6. Selezionare la scheda corrispondente al modello collegato: *Tools* > *Board* (ad esempio Arduino Uno).
7. Selezionare la porta USB utilizzata: *Tools* > *Serial Port* (se l'upload non dovesse funzionare selezionare un'altra porta seriale).
8. Compilare e caricare (*Upload*) su Arduino lo sketch mediante il pulsante (→): dovrebbero lampeggiare i LED RX e TX. Una volta finito l'uploading (messaggio: *Done uploading*) lo sketch caricato (*Blink*) dovrebbe far lampeggiare il LED *L* collegato al pin 13.
9. In caso di problemi selezionare: *Help* > *Troubleshooting*.
10. Caricare altri esempi premendo il pulsante *Open* (↑).

11. Per chiarimenti sulla sintassi del linguaggio di programmazione di Arduino selezionare: *Help* > *Reference*.

Caratteristiche della scheda ARDUINO UNO

- Microcontrollore: ATmega328 (Atmel)
- Tensione di lavoro: 5V
- Tensione d'ingresso (raccomandata): 7-12V (il + è il contatto interno del connettore)
- Digital I/O Pins: 14 (di cui 6 forniscono segnali d'uscita PWM)
- Analog Input Pins: 6
- DC Current per I/O Pin: 40 mA
- Flash Memory: 32 KB (ATmega328) di cui 0,5 KB usati dal bootloader
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Clock Speed: 16 MHz

Input e Output

Digital input/output (14 pin)

Ognuno dei 14 digital pins può essere usato come input o output, usando le istruzioni pinMode(), digitalWrite(), and digitalRead(). Essi operano tra 0 V e 5 V e possono fornire o ricevere al massimo 40 mA.

I digital pin hanno un resistore di pull-up interno (disconnesso per default) di 20-50 kΩ.

Per attivare il pull-up interno si usano queste istruzioni:

```
pinMode(pin, INPUT);      // predisponi pin come input
digitalWrite(pin, HIGH);  // attiva il resistore di pullup
```

In aggiunta alcuni digital pin hanno le seguenti **funzionalità speciali**:

- *Serial*: 0 (RX) and 1 (TX) usati per ricevere (RX) e trasmettere (TX) dati seriali TTL; sono connessi ai pins corrispondenti del ATmega8U2 USB-to-TTL Serial chip.
- *External Interrupts*: i pin 2 e 3 possono essere usati per fornire un interrupt in corrispondenza di un livello basso, un fronte di discesa o di salita, o un cambio di valore del segnale; vedere la funzione attachInterrupt().
- PWM: i pin 3, 5, 6, 9, 10, 11, quando sono impostati come uscite, possono fornire segnali PWM (256 valori diversi specificati con 8 bit) mediante la funzione analogWrite().
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK); questi pin consentono la comunicazione con bus SPI (*Serial Peripheral Interface*: seriale, sincrono full-duplex) mediante la SPI library.
- LED (13): sulla scheda è presente un LED connesso al digital pin 13.

Analog inputs (6 pin)

Le tensioni sui pin denominati A0 - A5, sono convertite in digitale con 10 bit di risoluzione (1024 valori differenti). Per default essi convertono valori analogici compresi tra 0 V e 5 V; mediante il pin AREF e l'istruzione analogReference() è possibile cambiare il fondoscala.

In aggiunta alcuni analog pins hanno **funzionalità speciali**:

- TWI: A4 (o SDA pin) e A5 (o SCL pin) supportano la comunicazione TWI mediante la Wire library. TWI (*Two-Wire Interface*), detto anche I²C (*Inter-Integrated Circuit*), è un bus seriale, multi-master, su due linee sbilanciate, per il collegamento di periferiche a bassa velocità a una scheda madre.

Sulla scheda sono presenti altri due pin:

- AREF: tensione di riferimento per gli ingressi analogici; da usare con l'istruzione analogReference().
- Reset: portato BASSO resetta il microcontrollore.

Sintassi del linguaggio C/C++ per Arduino

Il software in linguaggio C per Arduino si scrive nell'IDE (*Integrated Development Environment*) rappresentato in figura 3, che mette a disposizione gli strumenti per editare, salvare, compilare, caricare su Arduino, correggere (*debug*), ecc. il programma (detto *sketch*).
Si richiamano in forma sintetica le principali istruzioni per la programmazione di Arduino e alcuni esempi applicativi; si tenga presente che gli esempi riportati in seguito possono essere copiati e incollati nell'IDE Arduino.

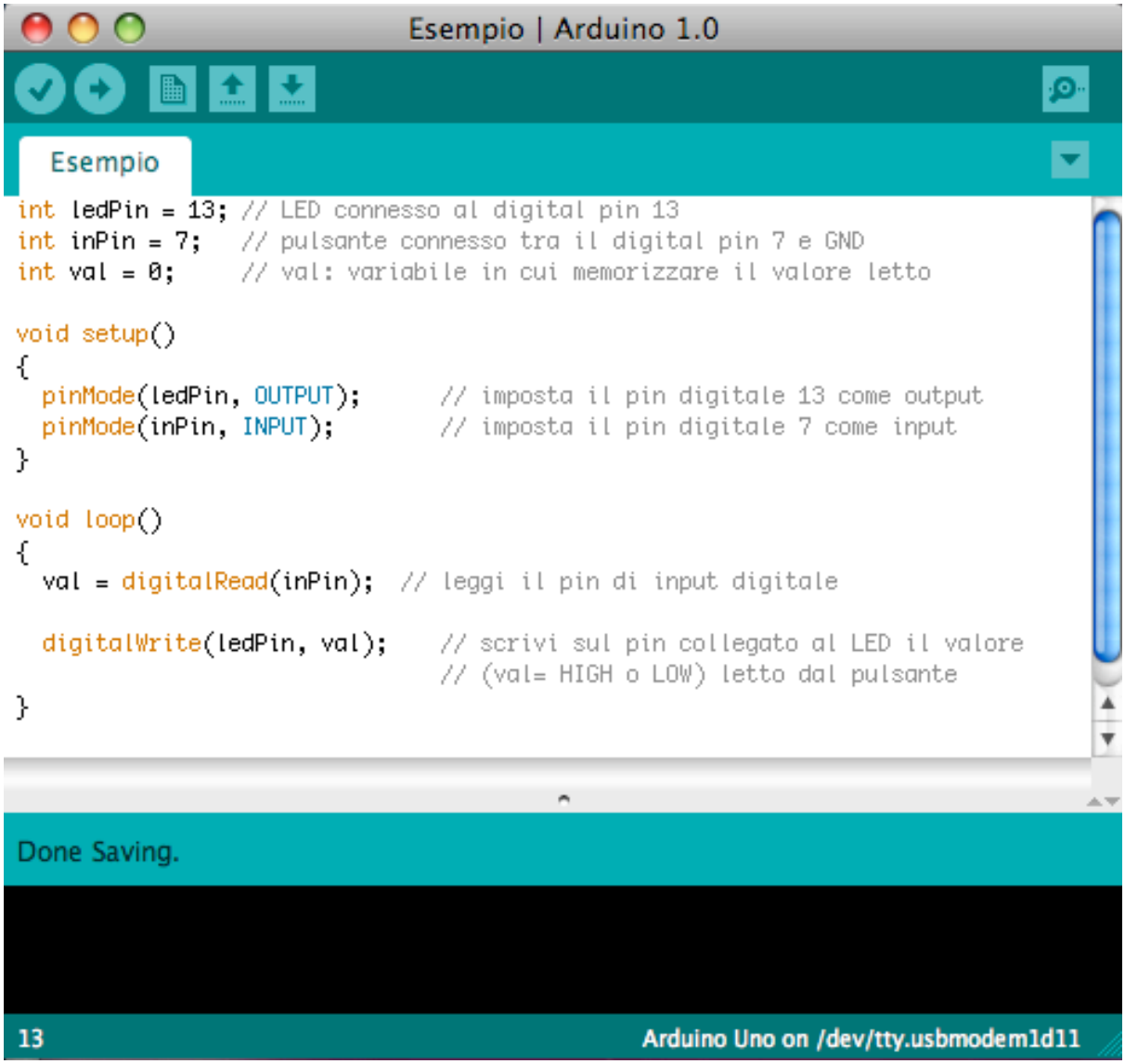

figura 3 - Esempio di programma (sketch) scritto nell'ambiente integrato di sviluppo (*IDE*) di Arduino.

Alcune informazioni generali sulla programmazione di Arduino:

- Per la sintassi del linguaggio di programmazione di Arduino seguire il link *reference* nella home page di Arduino oppure selezionare nell'IDE: *Help* > *Reference*.
- Per verificare la *correttezza sintattica* del programma scritto premere √ (*Verify*, in alto a sinistra nell'IDE).
- Tutte le istruzioni (tranne #define e #include <nome-libreria>) terminano con **;** (punto e virgola).
- // (commento singola linea)
- /*…*/ (commento linee multiple)
- per richiamare le parentesi graffe{} premere: su Windows: Alt+123; Alt+125; su Mac: shift+alt+[; shift+alt+].

Operatori aritmetici

- = (assegnazione: x=10), + (somma), - (differenza), x++ (incrementa x), x-- (decrementa x),
- (prodotto), / **(**quoziente), % (resto della divisione tra interi, es. x = 12 % 5; → x=2)

Operatori logici

- **&&** (and); || (or); **!** (not)
- operazioni bit a bit tra operandi con uguale n° di bit: & (and), | (or), ecc. (vedere *reference*)

Operatori di comparazione

x==y (x uguale a y), x!= y (x diverso da y), x< y, x> y , x<=y (x minore o uguale a y), x>=y

Principali tipi di variabili

Char: carattere ASCII, usa 1 byte (es.: char myChar = 'A';)
Int: intero, usa 2 byte, valori da -32.768 a 32.767 (es. int var = val;) (anche per variabili logiche)
Long: intero usa 4 byte, valori da -2.147.483.648 a 2.147.483.647
Float: usa 4 byte, valori da 3,4028235E+38 a -3,4028235E+38 (es. float sensorCalbrate = 1.117;)
Array: esempio di dichiarazione: **int myArray[10]={9,3,2,4,3,2,7,8,9,11};** vettore di 10 elementi numerati da 0 a 9, elencati tra graffe; richiamando **myArray[9]** si ottiene 11.

Struttura di un programma (sketch)

```
/*
Titolo
Descrizione
(commento di più righe)
 */

inserire QUI le definizioni. Esempi:
                #define ledPin 3 (il compilatore sostituisce il valore 3 a ledPin nel programma)
                int ledPin = 13;  (LED connesso al pin 13)
                #include <LiquidCrystal.h> (includi la libreria display LCD)

void setup() {

  inserire QUI il codice di SETUP che sarà eseguito 1 sola volta (viene mantenuto fino al
  caricamento di un altro sketch)
……………; //commento
……………; //commento
}

void loop() {
 inserire tra le graffe il codice del PROGRAMMA PRINCIPALE, che sarà eseguito
 ripetutamente (loop) finché la scheda non viene spenta o riprogrammata
……………;
……………; //commento
}

Inserire QUI eventuali FUNCTIONS richiamate nel programma

(fine dello sketch)
```

Esempi di semplici sketch per lo studio delle istruzioni fondamentali

Si riportano ora alcuni esempi con la descrizione delle principali istruzioni per la programmazione di Arduino. È possibile **copiare e incollare gli esempi riportati di seguito**, o porzioni di essi, nella schermata di programmazione del software Arduino (IDE).

Per effettuare il **debug del programma** si possono inviare dati da Arduino al monitor del computer, e viceversa, mediante le **istruzioni "Serial"**; per attivare il monitor premere il simbolo di ***Serial Monitor*** nell'IDE in alto a destra (figura 3):

```
Serial.begin(9600);  //imposta la velocità di comunicazione a 9600 bps
Serial.println(val);  // stampa il valore di val e vai a capo. Oppure dei caratteri: "Hello word"
Serial.read();  // restituisce il valore inviato mediante serial monitor
Serial.available();  // restituisce il numero di byte arrivati e disponibili per la lettura
flush();   // cancella la coda dei dati arrivati sulla porta seriale
```

Esempio:
impiego delle istruzioni "*Serial*"per inviare dati da Arduino al monitor del computer e viceversa; attivare il serial monitor col pulsante in alto a destra.

```
int incomingByte = 0;   // variabile per il dato in ingresso
void setup() {
        Serial.begin(9600);   // apri la porta seriale alla velocità di 9600 bps
}
void loop() {

        if (Serial.available() > 0) {      // invia i dati solo quando ricevi dei dati sulla porta
                incomingByte = Serial.read();  // leggi il byte in arrivo
                // scrivi il codice ASCII in decimale relativo al dato ricevuto:
                        Serial.print("I received: ");
                        Serial.println(incomingByte, DEC);
        }
}
```

Lettura e scrittura di valori digitali sui pin

```
/*
istruzioni: digitalRead(inPin) e digitalWrite(ledPin, val)
Il LED (pin 13) si accende e si spegne in base alla pressione di un pulsante (collegato al pin7).
I pin settati come INPUT possono essere collegati a Vcc mediante un resistore di pull-up interno
(vedi sopra: "per attivare il pull-up interno"); quindi in questo caso il pin scollegato esternamente è
interpretato come HIGH, mentre per portarlo LOW bisogna collegarlo a massa.
 */
int ledPin = 13; // LED connesso al digital pin 13
int inPin = 7;   // pulsante connesso tra il digital pin 7 e GND
int val = 0;     // val: variabile in cui memorizzare il valore letto

void setup()
{
  pinMode(ledPin, OUTPUT);      // imposta il pin digitale 13 come output
  pinMode(inPin, INPUT);      // imposta il pin digitale 7 come input
}

void loop()
{
  val = digitalRead(inPin);  // leggi il pin di input digitale (inPin collegato al pulsante)
                             // e restituisce 0 o 1
  digitalWrite(ledPin, val);    // scrivi sul pin collegato al LED il valore (val= HIGH o LOW)
                             // letto dal pulsante
}
```

Variante:
Dopo l'istruzione val = digital Read(inPin); inserire le istruzioni:

val=!val; // nega il valore di *val*
delay (1000); // attendi 1000 ms

e collegare tra loro i pin 7 e 13: in questo modo il led si accende e si spegne perché il valore sul pin 7 si inverte ad ogni lettura.

Lettura di valori analogici sui pin

```
/*
istruzione: analogRead(analogPin)
Converte la tensione d'ingresso compresa tra 0 V e 5 V presente su uno dei 6 pin di ingresso
analogico (A0-A5) in un numero intero compreso tra 0 e 1023 (intervallo di quantizzazione di 4,9
mV). Per modificare il campo usare analogReference(type) (dove ponendo type=EXTERNAL la
tensione applicata sul pin AREF pin (da 0 V a 5V) è usata come riferimento).
Istruzione: Serial.println(val) (comunicazione seriale col PC)
 */
int analogPin = 3;     // collegato al potenziometro (terminale centrale connesso all'analog pin 3
                       // e terminali esterni collegati a GND e +5V)
int val = 0;           // val: variabile in cui memorizzare il valore letto

void setup()
{
  Serial.begin(9600);     // inizializza la comunicazione seriale con il computer a 9600 bps
}

void loop()
{
  val = analogRead(analogPin);    // legge l'input pin e restituisce un intero compreso tra 0 e 1023
  Serial.println(val);            // visualizza i valori di val sul serial monitor del computer (premere il
                                  // pulsante serial monitor (in alto a destra nella finestra di Arduino)
}
```

Uscita analogica (PWM) sui pin digitali (3, 5, 6, 9, 10, 11)

```
/*
istruzione: analogWrite(pin, val)
Converte il valore val (0-255) in un segnale PWM (freq. 490 Hz) con duty-cycle variabile tra 0 % e
100%, su uno dei 6 digital pin (pin) PWM (3, 5, 6, 9, 10, 11).
Questo segnale può essere utilizzato per variare la luminosità di un LED, la velocità di un motore in
DC, la posizione di un servomotore, ecc.
Questo sketch modifica la luminosità di un LED (collegato al pin 9) modificando  la posizione di un
potenziometro (collegato al pin 3)
 */
int ledPin = 9;      // LED connesso al digital pin 9
int analogPin = 3;   // potenziometro connesso all'analog pin 3
int val = 0;         // variabile val dove memorizzare il valore letto

void setup()
{
  pinMode(ledPin, OUTPUT);   // imposta il pin come output
}

void loop()
{
  val = analogRead(analogPin);   // legge il pin di input
  analogWrite(ledPin, val / 4);  // analogRead: valori da 0 a 1023, analogWrite: valori da 0 a 255;
                                 // per questo val viene diviso per 4
}
```

Strutture di controllo del flusso del programma

Istruzione IF

```
if (espressione){
// blocco di istruzioni
}
```

Esegue il blocco d'istruzioni se l'espressione (*booleana*) è VERA, altrimenti passa oltre la graffa }.

Esempio con IF (accende i due LED se x>120):

```
if (x > 120){
  digitalWrite(LEDpin1, HIGH);
  digitalWrite(LEDpin2, HIGH);
}
```

Istruzione IF...ELSE

```
if (espressione)
{
  // blocco A
}
else
{
  // blocco B
}
```

Se *espressione* è VERA esegue il blocco di istruzioni A, altrimenti esegue il blocco B, poi passa oltre l'ultima graffa }.

```
/*
Esempio con l'istruzione IF
Lo sketch aumenta e diminuisce la luminosità (fade) di un LED collegato al pin 9.
 */
int brightness = 0;    // luminosità del LED inizializzata a 0
int fadeAmount = 5;    // incremento del fade a passi di 5

void setup()  {
  pinMode(9, OUTPUT);
}

void loop()  {
        analogWrite(9, brightness);   // imposta la luminosità del LED (variando il duty-cycle)
        brightness = brightness + fadeAmount; //aumenta o diminuisci la luminosità

                // inverti la direzione del fading alla fine del ciclo
        if (brightness == 0 || brightness == 255) {
          fadeAmount = -fadeAmount;
        }
        delay(30); // aspetta 30 ms per vedere l'effetto
}
```

Istruzione SWITCH …CASE

```
switch (var) {
   case 1:
     //blocco 1 di istruzioni quando var = 1
     break;
   case 2:
     // blocco 2 di istruzioni quando var = 2
     break;
   default:
     // se var non corrisponde ai casi elencati esegui queste istruzioni (default è opzionale)
  }
```

Esegue differenti blocchi di istruzioni, in base al valore (intero) della variabile *var*.

Istruzione **FOR**

```
for (inizializzazione; condizione; incremento) {
        //istruzioni
}
```

Ripete il ciclo tra le graffe a partire dalla condizione iniziale (*inizializzazione*), finchè la variabile di controllo, che a ogni ciclo viene incrementata della quantità *incremento*, soddisfa la condizione (*condition*). E' più potente di un FOR classico, perché *condizione* può essere un'espressione qualunque.

```
/*
Esempio per l'istruzione FOR
Lo sketch accende e spegne in sequenza i LED collegati ai pin 2-7.
 */
int timer = 100;          // tempo di ritardo 100 ms

void setup() {
  // usa un ciclo for per inizializzare i pin 2-7 come output
  for (int thisPin = 2; thisPin < 8; thisPin++)  {
    pinMode(thisPin, OUTPUT);
  }
}
void loop() {
          // loop dal più basso al più alto. thisPin++ incrementa thisPin
          for (int thisPin = 2; thisPin < 8; thisPin++) {
            digitalWrite(thisPin, HIGH);   // porta ON il pin
            delay(timer);
            digitalWrite(thisPin, LOW);    // porta OFF il pin
          }
          // loop dal più alto al più basso. thisPin-- decrementa thisPin
          for (int thisPin = 7; thisPin >= 2; thisPin--) {
            digitalWrite(thisPin, HIGH); // porta ON il pin
            delay(timer);
            digitalWrite(thisPin, LOW); // porta OFF il pin
          }
}
```

Altro esempio con ciclo FOR:

```
// variazione della luminosità di un LED
void loop()
{
  int x = 1;
  for (int i = 0; i > -1; i = i + x){
    analogWrite(PWMpin, i);
    if (i == 255) x = -1;          // inverti la direzione in corrispondenza degli estremi
    delay(10);
  }
}
```

Istruzione WHILE

```
while(expression){
  // blocco di istruzioni
}
```

Il blocco di istruzioni viene ripetuto finché l'espressione (booleana) è VERA (quando diventa falsa si procede con le istruzioni dopo la graffa **}**. L'espressione viene testata PRIMA dell'esecuzione del blocco di istruzioni.

Esempio con while:

```
val = digitalRead(inPin);
while(val){
  // blocco di istruzioni ripetuto finché val non diventa falso (inPin collegato a massa)
  val = digitalRead(inPin);
}
```

Istruzione DO…WHILE

```
do
{
   // blocco di istruzioni
} while (expression);
```

do…while è come while, ma la condizione è testata dopo aver eseguito il blocco di istruzioni.

Istruzione GOTO e label (etichetta)

Generalmente nella programmazione strutturata è un'istruzione da evitare; usata con cautela a volte può semplificare il programma.
label: // etichetta che individua una riga del programma
goto label; // trasferisce il flusso del programma all'etichetta *label*.

Esempio con GOTO:

```
for(byte r = 0; r < 255; r++){
   for(byte g = 255; g > -1; g--){
      for(byte b = 0; b < 255; b++){
         if (analogRead(0) > 250){ goto bailout;}
         // more statements ...
      }
   }
}
bailout:  // etichetta
```

Funzioni matematiche e trigonometriche

min(x, y) max(x,y) : restituiscono il minimo/massimo tra x e y;
abs(x) : valore assoluto di x;
sensVal = constrain(sensVal, 10, 150); // limita il range di sensVal tra 10 e 150;
pow(base, exponent) : eleva la base all'esponente;
sqrt(x) : calcola la radice quadrata di x;
sin(rad), cos(rad), tan(rad): sen, cos e tan dell'argomento in radianti (tipo float);
x = random(10, 20); // x assume un valore casuale compreso tra 10 e 19;
randomSeed(analogRead(0)); //inizializza il generatore casuale convertendo la tensione di rumore di un pin sconnesso (0), così tutte le volte che si avvia si ottengono valori diversi;
map(value, fromLow, fromHigh, toLow, toHigh) : cambia il range della variabile *val*;

Esempio con map:

```
void setup() {}

void loop()
{
  int val = analogRead(0);  // legge un valore analogico (e converte a 10 bit) dal pin 0
  val = map(val, 0, 1023, 0, 255); // cambia il range di val da 0-1023 al range 0-255
  analogWrite(9, val); // scrive val (8 bit) sul pin 9 (PWM)
}
```

Operazioni su bit e byte: lowByte(), highByte(), bitRead(), bitWrite(), bitSet(), bitClear(), bit()

Altre funzioni:

tone(pin, frequency) oppure tone(pin, frequency, duration): genera una nota musicale;
noTone(pin): ferma la nota;
millis(): restituisce il n° di ms dall'avvio del programma o dall'ultimo reset;
pulseIn(pin, value, timeout): restituisce la durata in µs (da 10 µs a 3 min) di un impulso sul *pin* specificato; se *value*=HIGH l'impulso inizia quando il pin va HIGH e viceversa; *timeout* (opzionale)=n° di µs (di default 1 s) entro cui deve iniziare l'impulso, altrimenti la funzione restituisce il valore 0;
random(min, max): genera un numero casuale compreso tra min e max; perché sia veramente casuale la funzione deve essere inizializzata con un valore, letto su un pin analogico scollegato (rumore), mediante l'istruzione randomSeed();
shiftOut(dataPin, clockPin, bitOrder, value): trasmette in seriale il byte *value* sul pin *dataPin*, usando il clock su *clockPin*, a partire dal MSB, ponendo *bitOrder*= MSBFIRST o viceversa;
shiftIn(dataPin, clockPin, bitOrder): funzione inversa di *shiftOut*.

FUNCTIONS

Le *functions* sono porzioni di codice che, quando richiamate, eseguono una funzione (es. calcolo della media) e restituiscono il risultato associato al nome della function; consentono di segmentare e compattare il programma in quanto si scrive il codice una sola volta, ma lo si può richiamare da vari punti del programma.
Due funzioni standard sono *setup*() e *loop*(), ma altre funzioni possono essere create esternamente alle graffe di queste due, ad esempio prima o dopo *loop*().
Se la funzione non restituisce alcun valore, il nome è preceduto da "**void**".

Esempio di sintassi: la *function* "moltiplica" (dichiarata fuori dal ciclo loop) esegue il **prodotto di due interi**:

```
int moltiplica(int x, int y){   // la funzione moltiplica elabora due interi x e y passati
                                // attraverso la chiamata e restituisce un intero
int risultato;
risultato =x*y;     // esegue il prodotto tra x e y
return risultato;       // restituisce la variabile risultato (che viene associata a moltiplica e
                        // che deve essere intera come moltiplica)
}
```

Per richiamare questa funzione all'interno del loop:

```
void loop{
        int i = 2;
        int j = 3;
        int k;

        k = moltiplica(i, j);   // chiamata alla function moltiplica; i e j sono passati alla
                                // funzione al posto di x e y;  moltiplica  restituisce il prodotto,
                                // k ora vale i*j=6
}
```

L'intero sketch con la function *moltiplica* è così:

```
void setup(){
  Serial.begin(9600);
}

void loop() {
  int i = 2;
  int j = 3;
  int k;

  k = moltiplica(i, j); // si assegna a k il valor restituito dalla function (ora contiene 6)
  Serial.println(k);
  delay(500);
}

int moltiplica(int x, int y){           // codice della function moltiplica
   int risultato;
```

```
  risultato = x * y;
  return risultato;
}
```

Altro esempio di function: creiamo (fuori dal ciclo loop) la funzione ***LeggiSens_e_Condition*** che legge un sensore 5 volte e calcola la media delle letture, scala il valore in 8 bit (da 0 a 255) e restituisce il valore invertito (255-sval):

```
int LeggiSens_e_Condition (){
  int i;
  int sval = 0;

  for (i = 0; i < 5; i++){
    sval = sval + analogRead(0);    // sensore sull'analog pin 0
  }

  sval = sval / 5;    // media
  sval = sval / 4;    // scala a 8 bits (0 - 255)
  sval = 255 - sval;  // inverti l'uscita
  return sval;
}
```

Per richiamare la funzione basta assegnarla ad una variabile dentro al loop():

```
void loop(){
        int sens;
        sens = LeggiSens_e_Condition ();
}
```

INTERRUPT

E' possibile interrompere l'esecuzione del programma e lanciare una routine di servizio quando si verifica un **interrupt esterno** (fornito sui pin 2 o 3). Per gli interrupt interni (forniti dal timer del microcontrollore) si veda la documentazione. Queste sono le istruzioni principali per gestire gli interrupt esterni:

- attachInterrupt(interrupt, function, mode): *interrupt* numero dell'interrupt [0 su digital pin 2, 1 su digital pin 3]; *function*: la funzione o la routine da richiamare quando si verifica l'interrupt (nessun parametro fornito o restituito); *mode* definisce la modalità di trigger (LOW quando il pin è low, CHANGE quando il pin cambia valore, RISING sul fronte di salita, FALLING sul fronte di discesa).
- detachInterrupt(interrupt): disabilita l'interrupt 0 o 1.
- interrupts(): riabilita gli interrupt disabilitati con l'istruzione noInterrupts().

// Esempio di utilizzo dell'interrupt

```
        int pin = 13;
        volatile int state = LOW;
```

```
void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

void loop()
{
  digitalWrite(pin, state);
}

void blink()   // function richiamata da attachInterrupt(0, blink, CHANGE)
{
  state = !state;
}
```

Alcune LIBRERIE

Le librerie mettono a disposizione delle istruzioni che consentono di eseguire in modo semplice alcune funzionalità extra, come il pilotaggio di display a cristalli liquidi, di motori passo-passo, di servomotori, ecc.
Per includere una libreria nello sketch si usa l'istruzione **#include<nome_libreria>**.

```
/*
  LiquidCrystal Library - Hello World

 Uso di un display LCD 16x2 (compatibile con Hitachi HD44780 ).
 Lo sketch scrive "Hello World!" sull'LCD e mostra i secondi dall'avvio.
[RS pin al digital pin 12, Enable pin 11, D4 al pin 5, D5 al pin 4, D6 al pin 3, D7 al pin 2, R/W pin
a GND, wiper al LCD VO pin (pin 3)]
 */

#include <LiquidCrystal.h> // include la libreria LiquidCrystal
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // inizializza la libreria con i numeri dei pin di interfaccia

void setup() {

  lcd.begin(16, 2);  // imposta il numero di colonne e righe del display
  lcd.print("hello, world!"); // scrive il messaggio "hello, world!" sul display
}

void loop() {

        lcd.setCursor(0, 1); // imposta il cursore su colonna 0, riga 1 (colonne 0-15, righe 0-1)

        lcd.print(millis()/1000); // scrive il numero di secondi dal reset
}
```

Variante del loop per far lampeggiare una scritta sul **display LCD**:

```
void loop() {

        lcd.noDisplay(); // spegni il display
        delay(500);

        lcd.display(); // accendi il display
        delay(500);
}
```

Altre istruzioni della libreria LiquidCrystal:
lcd.clear(); // cancella tutto il display
lcd.home(); // porta il cursore in alto a sinistra
lcd.cursor(); lcd.noCursor() // mostra/nascondi il cursore sul display; vedi anche lcd.blink()
lcd.scrollDisplayLeft(); // trasla il contenuto a sinistra di 1 posizione; lcd.scrollDisplayRight()
lcd.autoscroll(); // attiva l'autoscroll (i caratteri si aggiungono a destra traslando verso sinistra)
lcd.noAutoscroll(); // disattiva l'autoscroll
lcd.createChar(num, data); // crea carattere personalizzato (*gylph*) per l'LCD. Si possono definire al massimo 8 caratteri di 5x8 pixels (numerati da 0 a 7), ognuno specificato mediante un array di 8 bytes, uno per ogni riga del carattere; i 5 bit meno significativi di ogni byte determina i pixel di quella riga. Per visualizzare un carattere custom sul display: **write(n)** dove n è il numero del carattere.

//**Esempio** di realizzazione di un carattere (*smiley*)

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte smiley[8] = {      // definisci l'array di 8 elementi che specifica i pixel del carattere
  B00000,
  B10001,
  B00000,
  B00000,
  B10001,
  B01110,
  B00000,
};
void setup() {
  lcd.createChar(1, smiley);  // il carattere smiley è il numero 1
  lcd.begin(0, 0);   // scrivi in alto a sinistra
  lcd.write(1);       // la faccina sorridente
}
void loop() {}
```

SERVO

```
/*
Esempio di pilotaggio di un servomotore mediante la libreria "Servo"
Fa compiere ciclicamente al servo la massima escursione (0°-180°).
*/

#include <Servo.h>

Servo myservo;  // crea il servo object myservo per controllare un servomotore (massimo 8 oggetti)

int pos = 0;    // variabile pos per memorizzare la  posizione del servo

void setup()
{
  myservo.attach(9);  // collega il pin 9 al servo object myservo
}

void loop()
{
        for(pos = 0; pos < 180; pos += 1)  // escursione da  0° a 180° a passi di 1°
        {
          myservo.write(pos);           // manda il servo alla posizione pos
          delay(15);                    // attendi 15ms per far raggiungere la posizione al servo
        }
        for(pos = 180; pos>=1; pos-=1)     // escursione da  180° a 0° a passi di 1°
  {
    myservo.write(pos);             // manda il servo alla posizione pos
    delay(15);                      // attendi 15ms per far raggiungere la posizione al servo
  }
}
```

Variante: pilotare la posizione del servo con un potenziometro

```
#include <Servo.h>
Servo myservo;

int potpin = 0;  // analog pin usato per connettere il potenziometro
int val;    // variabile val per leggere il valore del pin analogico

void setup()
{
  myservo.attach(9); // servo collegato al pin 9
}

void loop()
{
        val = analogRead(potpin);            // legge il valore del potenziometro (tra 0 e 1023)
        val = map(val, 0, 1023, 0, 179);     // converte il valore di val in una scala compresa
```

```
                                        // tra 0 e 180 (posizione del servo in gradi)

        myservo.write(val);             // invia al servo la posizione da raggiungere
        delay(15);                      // aspetta che il servo arrivi a destinazione
}


/* Variante: la posizione del servo viene incrementata o decrementata in base
al valore digitale posto sul pin 2
*/

#include <Servo.h>

Servo myservo;  // crea il servo object myservo per controllare un servomotore (al massimo 8)

int pos = 90;    // variabile pos per memorizzare la  posizione del servo
int verso;

void setup()
{
  myservo.attach(9);  // collega il pin 9 al servo object myservo
  pinMode(2, INPUT);
}

void loop()
{
        verso = digitalRead(2);
        if(verso=0)
        {
          myservo.write(pos);       // manda il servo alla posizione pos
          pos = pos + 1;            // incrementa la posizione
          delay(35);                // attendi 15ms per far raggiungere la posizione al servo
        }
        else
        {
          myservo.write(pos);       // manda il servo alla posizione pos
          pos = pos - 1;            // decrementa la posizione
          delay(35);                // attendi 15ms per far raggiungere la posizione al servo
        }
}
```

Altre librerie

- SoftwareSerial: consente la comunicazione seriale su ogni piedino digitale, con velocità fino a 115200 bps;
- Stepper: pilota motori passo-passo unipolari e bipolari
- SD: legge e scrive le SD cards
- XBee: per comunicazioni wireless con il modulo XBee
- SimpleTimer(): consente l'esecuzione di una porzione di codice ogni *n* millisecondi, senza l'impiego di interrupt e del timer interno.

Sul sito di Arduino (*Reference*> *Libraries*) sono disponibili molte altre librerie con le istruzioni per il loro utilizzo.

