La programmazione di Arduino

Si analizza la sintassi delle principali istruzioni per Arduino mediante alcuni esempi applicativi (*sketch*); per la sintassi completa del linguaggio di programmazione di Arduino seguire il link *reference* nella home page di Arduino (*www.arduino.cc*) oppure selezionare nell'IDE: Help > Reference.
E' 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 ***Serial Monitor*** in alto a destra):

```
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: riceve dal monitor e scrive sul monitor (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).
Alcuni pin settati come INPUT possono essere collegati a Vcc mediante un resistore di pull-up
interno (si veda nel testo: "per attivare il pull-up interno"); in questo caso il pin scollegato
esternamente è interpretato come HIGH, mentre per portarlo LOW bisogna collegarlo a massa, ad
esempio mediante un pulsante. Non attivando il pull-up interno (come nell'esempio presente) è
necessario impiegare un pull-up esterno.
 */
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 sui pin analog in

```
/*
```

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 (0 to 5V) è usata come riferimento).
Istruzione: **Serial.println(val)** per la 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 analogPin e restituisce un intero 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 (di frequenza 490 Hz) con duty-cycle variabile tra 0 % e 100%, su uno dei 6 pin digital 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 come partitore di tensione 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 l'input analogPin e restituisce un intero tra 0 e 1023
  analogWrite(ledPin, val / 4);  // fornisce al pin 9 (ledPin) un segnale PWM
                                 // poiché val è compreso tra 0 e 1023 il suo valore viene diviso
                                 // per 4, in quanto analogWrite accetta valori tra 0 e 255
}
```

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 si esegue il blocco di istruzioni A, altrimenti si esegue il blocco B.

```
/*
Esempio con l'istruzione IF

Lo sketch varia la luminosità (fade) di un LED sul pin 9, up e down
 */
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 osservare 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 blocchi differenti 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*), fino a quando la variabile di controllo, che ad 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 funziona come *while*, ma la condizione è testata dopo aver eseguito il blocco di istruzioni.

Istruzione: **GOTO**

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;}
         // altre istruzioni ...
      }
   }
}
bailout:  // etichetta
```

Funzioni matematiche e trigonometriche

min(x,y) e 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;
Operazioni su bit e byte: lowByte(), highByte(), bitRead(), bitWrite(), bitSet(), bitClear(), bit()

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)
}
```

Altre funzioni:

tone(pin, frequency) oppure tone(pin, frequency, duration): genera nota musicale
noTone(pin): blocca la nota musicale
millis(): restituisce il n° di millisecondi trascorsi dall'avvio del programma
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" esegue il **prodotto di due interi** (dichiarata fuori dal ciclo loop):

```
int moltiplica(int x, int y){   // la funzione moltiplica elabora due interi x e y passati
                                // attraverso la chiamata e restituisce un intero
int result;
result=x*y;     // esegue il prodotto tra x e y
return result;          // restituisce la variabile result (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 result;
  result = x * y;
  return result;
}
```

Altro esempio di function**:** creiamo, fuori dal ciclo *loop*, la funzione *ReadSens_and_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 (complementare a 255, cioè 255-*sval*):

```
int ReadSens_and_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 bit (valori da 0 a 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 = ReadSens_and_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 sul sito Arduino. Queste sono le istruzioni principali per gestire gli interrupt esterni:

- attachInterrupt(interrupt, function, mode): *interrupt* è il numero dell'interrupt [0 su digital pin 2, 1 su digital pin 3]; *function* è 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;
}
```

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>**.

Libreria LiquidCrystal

```
/*
 Uso di un display LCD 16x2 (compatibile con Hitachi HD44780 ).
 Lo sketch scrive "Hello World!" sull'LCD e mostra i secondi dall'avvio.
[RS collegato al digital pin 12, Enable al 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 pin LCD VO (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!"); // stampa 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); // stampa il numero di secondi dal reset
}
```

Variante del loop per far lampeggiare la 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(); // trasla il contenuto a destra di 1 posizione;
lcd.autoscroll(); // attiva l'autoscroll (i caratteri si aggiungono a destra traslando verso sinistra)
lcd.noAutoscroll(); // disattiva l'autoscroll
lcd.createChar(num, data); // crea un 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 determinano 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 (bytes) 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() {}
```

Libreria 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 (max 8)
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 mediante 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 disponibili

- 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 mediante il modulo XBee
- SimpleTimer(): consente l'esecuzione di una porzione di codice ogni *n* millisecondi, senza l'impiego di interrupt e del timer interno.