Appendice
La sintassi di Arduino

1 Informazioni generali

Ricordiamo alcune informazioni generali sulla programmazione di Arduino.

- Per accedere alla sintassi completa del linguaggio di programmazione di Arduino, seguire nell'IDE il percorso: *Aiuto* → *Guida di riferimento* o andare sul sito www.arduino.cc e selezionare *Resources* → *Reference*.
- Per verificare la correttezza sintattica del programma scritto premere (*verifica*) in alto a sinistra nell'IDE.
- Tutte le istruzioni (tranne `#define` e `#include <nome-libreria>`) terminano con `;` (punto e virgola);
- I commenti si inseriscono in questo modo:

```
// ............ testo del commento .....................
(commento su linea singola)
/*............ testo del commento ...................*/
(commento su linee multiple)
```

- Per scrivere le parentesi graffe {} digitare:

```
su Windows: Alt+123 e Alt+125 oppure Alt-Gr+Shift+[ e
            Alt-Gr+Shift+]
su Mac:     Shift+alt+[  e Shift+alt+]
```

2 Struttura dello sketch

Uno sketch è costituito dalle seguenti sezioni:

1. **Titolo e descrizione** (facoltativi ma consigliati).
   Sono preceduti da `/*` e seguiti da `*/`.
   Esempio

```
/*
Titolo e descrizione
*/
```

2. **Dichiarazioni direttive**
   – Dichiarazioni di costanti e variabili globali.
   Esempio

```
const int pinLed = 13;// dichiarazione della costante
                      // pinLed
int val;              // dichiarazione della variabile val
```

– Le **direttive delle librerie** incluse nel programma, precedute da `#include`.

Esempio

```
#include <Ethernet.h> // include la libreria Ethernet
                      // per collegarsi in rete tramite
                      // lo shield Ethernet
```

– Le direttive `#define...` a cui sintassi è (la direttiva *non* termina con ; e *non* vuole = tra nome e valore):

```
#define nomediunacostante valore
```

Con questa direttiva si indica al compilatore di sostituire nello sketch, in fase di compilazione, il valore specificato al nome della costante definita.

Per esempio `#define` può essere impiegata per identificare con un nome significativo uno dei pin di Arduino:

```
#define pin_LED 7  // denomina pin_LED il pin n. 7 a cui è
                   // collegato un LED
```

Invece è preferibile utilizzare la dichiarazione **`const`** per assegnare un nome a una generica costante utilizzata in un programma.

3. **Il codice di SETUP** (che sarà eseguito una sola volta), preceduto da `void setup(){`.

Esempio

```
void setup() {
  Serial.begin(9600);       // inizializza il monitor seriale
  pinMode(pin_LED,OUTPUT); // inizializza il pin denominato
                           // (con #define) pin_LED come
                           // uscita
}
```

4. **Il codice del programma principale** (che sarà eseguito ripetutamente finché la scheda non viene spenta, resettata o riprogrammata), preceduto da `void loop(){`.

Esempio

```
void loop() {
  digitalWrite(pin_LED, HIGH); // accendi il LED
  delay(1000);                 // aspetta un secondo
  digitalWrite(pin_LED, LOW);  // spegni il LED
  delay(1000);                 // aspetta un secondo
}
```

5. Le eventuali *funzioni* (**FUNCTION**) richiamate nel programma, dove con il termine *funzione* si intende un blocco di istruzioni che viene eseguito quando la funzione, con i valori degli eventuali parametri, viene richiamata nel corpo dello sketch. La sintassi con cui si dichiara una funzione è la seguente:

```
tipo nome_funzione(parametro1, parametro2, ..){
istruzione 1;
istruzione 2;
.....
}
```

Se la funzione deve restituire un valore conterrà una variabile da restituire e terminerà con l'istruzione `return` *nomevariabile*, mentre se la funzione non deve restituisce alcun valore, il nome della function è preceduto da `void` e manca l'istruzione `return`.
Due funzioni standard sono `setup()` e `loop()`, ma si possono creare altre funzioni esternamente alle graffe di queste due, per esempio dopo `loop()` oppure prima di `setup()`.

ESEMPIO

La function *moltiplica* (dichiarata fuori dal ciclo `loop`) esegue il prodotto di due interi; riceve i due fattori (*a* e *b*) che associa ai due parametri *x* e *y*, e restituisce un intero (prodotto) associato al nome della function *moltiplica*. Quindi il risultato è: $p = a \cdot b = 6$.

```
void loop() {
    int a = 2;
    int b = 3;
    int p = moltiplica(a, b);   // 1) chiama la function "moltiplica", passando i valori
                                // dei parametri a, b;
                                // 2) la function restituisce la variabile
                                // prodotto = x * y = 6;
                                // 3) p assume quindi il valore 6
}

int moltiplica(int x, int y){   // codice della function "moltiplica",
                                // x, y assumono i valori dei parametri a, b, passati
                                // nella chiamata
  int prodotto = x * y;         // esegui il prodotto
  return prodotto;              // la function "moltiplica" restituisce
                                // la variabile "prodotto"
}
```

3 Operatori

Operatori aritmetici

=	assegnazione (x=10)
+	somma
-	differenza
x++	incrementa x
x--	decrementa x
*	prodotto
/	quoziente
%	resto della divisione tra interi

Esempio

x = 12 % 5 ⟶ *x* vale 2

Operatori logici

&&	AND
||	OR
!	NOT

Operatori di comparazione

x==y	x uguale a y
x!=y	x diverso da y
x<y	x minore di y
x>y	x maggiore di y
x<=y	x minore o uguale a y
x>=y	x maggiore o uguale a y

4 Tipi di dati

`const` costante.

`char` carattere ASCII, usa 1 byte.

Esempio

```
char myChar = 'A'
```

`bool` variabile logica (true=HIGH=1, false=LOW=0). Usa 1 byte.

`int` intero, valori da –32768 a +32767. Usa 2 byte.

`long` intero, valori da –2147483648 a +2147483647. Usa 4 byte.

`float` numero con virgola, valori da -3.4028235E+38 a +3.4028235E+38. Usa 4 byte.

Esempio

```
const float Pi = 3.14
```

`array` insieme di elementi omogenei, individuati da un indice numerico.

Esempio

```
int myArray[10]={19, 32, 2, 41, 38, 22, 7, 87, 9, 11}
```

dichiara un array di 10 elementi interi, numerati da 0 a 9, elencati tra le graffe; per esempio, richiamando `myArray[9]` si ottiene il numero 11.

`volatile` vanno dichiarate *volatili* le variabili modificate all'interno di una routine di servizio a un *interrupt*.

5 Monitor seriale

Per effettuare il *debug* del programma, durante l'esecuzione si possono inviare dati da Arduino al PC e viceversa, mediante le istruzioni Serial. Per attivare il Monitor Seriale premere l'icona  in alto a destra nell'IDE.

```
Serial.begin(9600)   // (nel setup) imposta la velocità di
                     // comunicazione a 9600 bps
```

Comunicazione Arduino → PC

```
Serial.println(val)  // scrive sul Monitor Seriale il valore
                     // della variabile val oppure caratteri
                     // tra apici come «Hello world»
                     // e va a capo
```

```
Serial.print(val)  // scrive sul Monitor Seriale senza andare
                   // a capo
```

Comunicazione PC → Arduino

```
Serial.available() // restituisce il numero di byte arrivati
                   // dal PC e disponibili per la lettura
Serial.read()      // legge e restituisce il valore inviato
                   // inviato dal PC mediante Monitor Seriale
flush()            // cancella la coda dei dati arrivati
                   // sulla porta seriale
```

6 Pin digitali e analogici

Pin digitali (0-13): lettura e scrittura

Nel setup

```
pinMode(pin, OUTPUT)          // imposta il pin come output
pinMode(pin, INPUT)           // imposta il pin come input
pinMode(pin, INPUT_PULLUP)    // imposta il pin come input
                              // attivando il pull-up interno
```

Istruzioni di lettura e scrittura

```
digitalRead(pin)       // legge il valore logico (HIGH o LOW)
                       // dal pin di input digitale (0-13)
digitalWrite(pin, val) // scrive sul pin di output digitale
                       // il valore val (HIGH o LOW)
```

In caso di necessità si possono usare come digitali anche i pin analogici, facendo riferimento agli indirizzi da 14 a 19.

ESEMPIO

Scrivi sul pin digitale 13 il valore letto da un pulsante collegato al pin digitale 7, dopo averlo negato.

```
void setup(){
  pinMode(13, OUTPUT);       // imposta il pin digitale 13 come output
  pinMode(7, INPUT_PULLUP);  // imposta il pin digitale 7 come input (pull-up interno)
}
void loop(){
  bool val = digitalRead(7); // leggi il valore sul pin di input digitale 7 e associalo a val
  val=!val;                  // nega il valore di val
  digitalWrite(13, val);     // scrivi sul pin 13 il valore val (HIGH o LOW)
}
```

Tutto il loop si può compattare in una riga unica equivalente:

```
digitalWrite(13, !digitalRead(7));
```

Pin analogici (A0-A5): lettura

```
analogRead(analogPin)       // legge la tensione su un pin
                            // analogico (A0-A5) e converte
                            // il valore compreso tra 0 V e
                            // 5 V in un numero intero
                            // compreso tra 0 e 1023
analogReference(EXTERNAL)   // (nel setup) modifica la tensione
                            // di fondo scala (5 V) portandola
```

```
                    // al valore della tensione Vref
                    // collegata al pin AREF
```

ESEMPIO Leggi ogni 2 secondi il valore di tensione sul pin A0 (fornita mediante un potenziometro) e scrivilo sul Monitor Seriale.

```
int analogPin = A0;       // pin A0 collegato al potenziometro
int val;                  // val: variabile in cui memorizzare il valore letto su A0

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

void loop(){
  val = analogRead(analogPin);  // associa a val il valore digitale (0-1023) corrispondente
                                // alla tensione letta sul pin A0
  Serial.println(val);          // scrivi il valore di val sul Monitor Seriale
}
```

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

```
analogWrite(pin, val)  // su uno dei 6 pin digital PWM
                       // (3, 5, 6, 9, 10, 11) converte
                       // il valore val (0-255) in un segnale
                       // PWM (frequenza 490 Hz)
                       // con duty-cycle (0% - 100%)
                       // proporzionale a val
```

ESEMPIO Modifica la luminosità di un LED (collegato al digital pin PWM 9), mediante un potenziometro (collegato all'analog pin 0).

```
int pinLed = 9;      // LED connesso al digital pin 9 (PWM)
int analogPin = A0;  // potenziometro connesso all'analog pin 0
int val = 0;         // variabile dove memorizzare il valore letto dal pin analogico

void setup(){
  pinMode(pinLed, OUTPUT);   // imposta il pin 9 come output
}

void loop(){
  val = analogRead(analogPin);    // legge la tensione sull'analog pin 0 e associa a val
                                  // il corrispondente valore digitale (0 - 1023)
  val= map(val, 0, 1023, 0, 255); // map converte il valore di val (0 - 1023) in un nuovo valore
                                  // compreso tra 0 e 255 (per l'istruzione analogWrite che segue)

   analogWrite(pinLed, val);      // genera un segnale PWM sul pin 9, con duty-cycle
                                  // proporzionale a val (0 - 255);
                                  // quindi la luminosità del LED aumenta all'aumentare di val
}
```

7 Strutture di controllo del flusso di programma

Istruzione FOR

```
for (inizializzazione; condizione; incremento) {
.........................; // blocco di istruzioni (ciclo)
}
```

Il FOR ripete il blocco di istruzioni tra le graffe finché la variabile di controllo (per esempio *x*), definita e inizializzata in inizializzazione e che a ogni ciclo viene incrementata della quantità incremento, soddisfa la con-

dizione (per esempio x <= 10). Quando la variabile non soddisfa più la condizione, si esce dal FOR e l'esecuzione del programma riprende dall'istruzione posta dopo la parentesi graffa di chiusura del ciclo.

ESEMPIO Varia la luminosità di un LED dal minimo al massimo

```
for (int i=0; i<=255; i++){     // per i che va da 0 a 255, a incrementi di 1 (i++)
      analogWrite(pinLed, i);   // invia al LED un segnale PWM di valore i
      delay(10);                // ad ogni ciclo fai una pausa di 10 ms
   }
```

Istruzione IF

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

Se l'espressione (booleana) è VERA, esegue il blocco di istruzioni; altrimenti passa oltre la parentesi graffa di chiusura del ciclo.

Istruzione IF...ELSE

```
if (espressione){
.............................;   // blocco di istruzioni A
}
else{
.............................;   // blocco di istruzioni B
}
```

Se l'espressione è VERA, esegue il blocco di istruzioni A; altrimenti il blocco B.

ESEMPIO Se x<100 incrementa x e y, altrimenti (se x >= 100) decrementa x e y.

```
if (x<100){   // se x<100
   x++;       // incrementa x e y
   y++;
}
else{         // altrimenti (x>=100)
   x--;       // decrementa x e y
   y--;
}
```

Istruzione SWITCH...CASE

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

Esegue blocchi differenti di istruzioni, in base al valore (1, 2, 3, ...) della variabile var.

Istruzione WHILE

```
while (espressione){
.............;   // blocco di istruzioni
}
```

Ripete il blocco di istruzioni finché l'espressione (booleana) è VERA; quando diventa FALSA l'esecuzione procede con le istruzioni dopo la parentesi graffa di chiusura. L'espressione viene testata prima dell'esecuzione del blocco di istruzioni.

ESEMPIO Ripeti un blocco di istruzioni finché non viene premuto un pulsante.

```
Puls = digitalRead(pinPuls);     // leggi lo stato del pulsante
while(Puls){
  ...       // blocco di istruzioni, ripetuto finché Puls non diventa FALSO
  ...
  Puls = digitalRead(pinPuls);   // aggiorna Puls
}
```

Istruzione DO...WHILE

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

DO...WHILE è analoga a WHILE, ma l'espressione è testata dopo aver eseguito il blocco di istruzioni; quindi il blocco viene eseguito almeno una volta.

ESEMPIO Leggi il pin 7, collegato al pulsante PA, finché questo non viene premuto.

```
do{
  PA = digitalRead(7);      // leggi il pulsante PA
}while(PA);                 // finché PA=1 (non premuto)
```

Istruzione BREAK

BREAK è usata per uscire da cicli FOR, WHILE e DO...WHILE, bypassando la normale condizione del ciclo.

Nell'esempio che segue, il servomotore viene portato da 0° a 180° a passi di 1°, con un ciclo FOR; quando si preme il pulsante durante il ciclo, il servomotore torna a 0° e si esce dal FOR, a causa dell'istruzione `break`.

```
for (pos = 0; pos <= 180; pos++){ // incrementa pos da 0 a 180
  servo1.write(pos);              // porta il servo1 a pos (gradi)
  delay(15);
  puls=digitalRead(pinPuls);      // leggi lo stato del pulsante
  if(puls==0){                    // se il pulsante è premuto
    pos = 0;                        // azzera pos
    servo1.write(pos);              // riporta il servo1 a zero
    break;                          // esci dal ciclo FOR
  }
}
```

Funzioni matematiche e trigonometriche

```
min(x, y)                       // restituisce il minimo tra
                                // x e y
max(x, y)                       // restituisce il massimo tra
                                // x e y
val = constrain(val, 10, 150)   // limita il range della
                                // variabile val tra 10 e 150
abs(x)                          // restituisce il valore
                                // assoluto di x
pow(base, exponent)             // eleva la base all'esponente
sin(rad)                        // seno dell'argomento rad in
                                // radianti
cos(rad)                        // coseno dell'argomento rad
                                // in radianti
tan(rad)                        // tangente dell'argomento rad
                                // in radianti
memcpy(array1, array2, 16)      // copia tutto l'array2
                                // sull'array1 (16 elementi)
sqrt(x)                         // calcola la radice quadrata
                                // di x
```

Altre funzioni

```
tone(pin, frequenza, durata)    // genera una nota musicale
                                // su un pin con una certa
                                // frequenza e durata in ms
                                // (durata è facoltativa)
noTone(pin)                     // ferma la nota sul pin
millis()                        // restituisce i millisecondi
                                // dall'avvio dello sketch
                                // 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 se LOW è il
                                // contrario;timeout
                                // (opzionale): n° di µs
                                // (di default 1 s) entro cui
                                // deve iniziare l'impulso
                                // o 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 nel setup()
                                // con l'istruzione
                                // randomSeed(analogRead(0))
val = map(val, 0, 1023, 0, 255) // map converte il valore di
                                // val (0 - 1023) in un nuovo
                                // valore compreso tra 0 e 255
```

8 Interrupt (esterni)

Quando si verifica un *interrupt* esterno (sui *digital pin* 2 o 3), viene interrotta l'esecuzione dello sketch e lanciata la relativa function di servizio. L'istruzione principale per la gestione degli interrupt è:

```
attachInterrupt(interrupt, function, mode)
```

dove:

```
interrupt  // numero dell'interrupt (0 su digital pin 2 1 su
           // digital pin 3);
function   // nome della funzione da richiamare quando si
           // verifica l'interrupt (nessun parametro fornito
           // o restituito);
mode       // modalità di interrupt: LOW quando il pin è low
           // CHANGE quando il pin cambia valore RISING
           // sul fronte di salita FALLING sul fronte
           // di discesa.
```

Le variabili modificate all'interno della function di servizio dell'interrupt devono essere dichiarate di tipo volatile.

ESEMPIO Ogni secondo scrivi sul *digital pin* 13 (LED a bordo della scheda) il valore della variabile state, che si inverte ogni volta che arriva un *interrupt* sul *digital pin* 2 (fronte di discesa: HIGH → LOW)

```
volatile int state = LOW;   // "state" (volatile) viene modificata dentro la function "blink"
                            // di servizio all'interrupt

void setup(){
  pinMode(13, OUTPUT);
  attachInterrupt(0, blink, FALLING);  // interrupt 0: quando il pin 2 passa da HIGH a LOW,
                                       // chiama la function "blink"
}

void loop(){
  digitalWrite(13, state);  // scrivi il valore di "state"
  delay(1000);              // ogni secondo
}

void blink(){               // function "blink" richiamata dall'interrupt sul pin 2
                            // non riceve parametri e non restituisce valori
  state = !state;           // inverti il valore di "state"
}
```

9 Librerie standard

Le librerie mettono a disposizione delle istruzioni che consentono di eseguire in modo semplice alcune funzionalità, come il pilotaggio di display LCD, di servomotori e di motori passo-passo. Per includere una libreria nello sketch, prima del setup si scrive la direttiva:

```
#include<nome_libreria>
```

<LiquidCrystal.h>

`<LiquidCrystal.h>` è la libreria per il pilotaggio di display LCD.

- Prima di `setup()`:

```
#include <LiquidCrystal.h>       // include la libreria
                                 // LiquidCrystal
LiquidCrystal lcd(12,11,5,4,3,2) // inizializza la libreria
                                 // con i numeri dei 6 pin
                                 // di interfaccia.
                                 // Per usare altri pin,
                                 // basta dichiararli nello
                                 // stesso ordine tra
                                 // le parentesi.
```

- Nel `setup()`:

```
lcd.begin(16, 2)                 // imposta il numero
                                 // di colonne (16) e righe
                                 // (2) del display
```

- Istruzioni principali:

```
lcd.setCursor(colonna, riga)     // porta il cursore nella
                                 // posizione del display
                                 // specificata da (colonna
                                 // 0-15, riga 0-1)
lcd.print(dato)                  // scrive il dato
                                 // (di tipo: char, byte,
                                 // int, long, string) sul
                                 // display; per esempio:
                                 // lcd.print(cont),
                                 // scrive nella posizione
                                 // del cursore il
                                 // valore della
                                 /  variabile cont, mentre
                                 // cd.print("Ciao!")
                                 // scrive la stringa "Ciao!"
lcd.clear()                      // cancella tutto il
                                 // display
lcd.noDisplay()                  // spegne il display
lcd.display()                    // riaccende il display
```

- Altre istruzioni della libreria:

```
lcd.home()                       // porta il cursore
                                 // in alto a sinistra
lcd.cursor()                     // mostra il cursore sul
                                 // display
lcd.noCursor()                   // nascondi il cursore sul
                                 // display
lcd.scrollDisplayLeft()          // trasla il contenuto a
                                 // sinistra di una posizione;
lcd.scrollDisplayRight()         // trasla il contenuto
                                 // a destra di una 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 (glyph)
                                 // per l'LCD (esempio 2)
```

```
lcd.write(data)                  // visualizza un carattere
                                 // personalizzato
                                 // sul display LCD
```

<Servo.h>

<Servo.h> è la libreria per pilotaggio di servomotori.

- Prima di `setup()`:

```
#include <Servo.h>   // include la libreria Servo.h
Servo servo1;        // servo1: nome assegnato al servomotore
```

- Nel `setup()`:

```
servo1.attach(pin); // specifica il pin a cui è collegato
                    // servo1
```

- Nel `loop()`:

```
servo1.write(val);  // invia a servo1 l'angolo
                    // da raggiungere (in gradi)
```

Altre librerie standard

Elenchiamo infine alcune librerie standard.

Stepper	pilota motori passo-passo unipolari e bipolari.
SD	legge e scrive le SD card.
XBee	gestisce 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.
SoftwareSerial	consente la comunicazione su porta seriale con ogni piedino digitale (velocità fino a 115200 bps).
EEPROM	scrive e legge nella memoria non volatile (EEPROM) del microcontrollore.
Ethernet	gestisce il collegamento a internet mediante la Ethernet Shield.
WiFi	gestisce il collegamento a internet mediante la Wifi Shield.
GSM	gestisce il collegamento alla rete GSM mediante la GSM Shield.
SPI	gestisce la comunicazione verso delle periferiche tramite una connessione seriale di tipo SPI (Serial Peripheral Interface)