MINI-GUIDA PER ARDUINO

Arduino

Arduino è una **scheda elettronica programmabile** che permette di controllare luci, motori, sensori e molti altri componenti in modo semplice.
È basata su un **microcontrollore**, cioè un piccolo "cervello" capace di leggere informazioni dall'ambiente (ad esempio la temperatura, la distanza, un pulsante) e prendere decisioni in base al codice che scriviamo.

A differenza di un normale computer, Arduino esegue un unico programma in modo continuo, senza sistema operativo e senza applicazioni: fa solo quello che gli diciamo di fare, ma lo fa **in tempo reale**, in modo affidabile e con consumi minimi.

Struttura base di uno sketch

Come si scrive un programma semplice

Ogni programma Arduino (chiamato *sketch*) contiene **due funzioni obbligatorie**:

- **setup()** - Viene eseguita una sola volta, subito dopo l'accensione o il reset della scheda.
  Qui si configurano i pin, si stabilisce la comunicazione seriale, si inizializzano variabili, sensori o moduli esterni. È il punto in cui "prepariamo" l'hardware a funzionare.
- **loop()** - Viene eseguita in un ciclo senza fine. Qui mettiamo la logica operativa: leggere un sensore, controllare un LED, calcolare valori, prendere decisioni, comunicare con altri dispositivi. Tutto ciò che la scheda deve fare ripetutamente va scritto qui.

```
void setup() {
      // Codice che viene eseguito UNA sola volta all'avvio
}

void loop() {
      // Codice che viene eseguito ALL'INFINITO
}
```

Variabili

Le variabili servono per **memorizzare valori** che il programma deve usare o modificare mentre funziona.
Ogni variabile ha un **tipo**, cioè indica che tipo di dato può contenere.

```
int numero = 5;             // numero intero (…, -2, -1, 0, 1, …)
float temperatura = 21.3;   // numero con la virgola
bool acceso = true;         // valore logico: true o false
String nome = "Carlo";      // testo (sequenza di caratteri)
```

Le variabili permettono ad Arduino di:

- memorizzare letture dai sensori (es. distanza, temperatura…)
- ricordare stati (es. "il LED è acceso?", "il tasto è premuto?")

- fare calcoli (es. media, somma, differenze)
- salvare testi da inviare via Serial Monitor

Sono la base di qualsiasi programma: senza variabili un robot non potrebbe prendere decisioni né ricordare cosa è successo un attimo prima.

Funzioni fondamentali

Per programmare Arduino è importante conoscere alcune funzioni base che permettono di leggere sensori, controllare attuatori e gestire il tempo. Queste funzioni sono già incluse nell'ambiente Arduino e possono essere usate **senza** librerie aggiuntive.

pinMode() Serve a scegliere se un pin deve funzionare come ingresso o come uscita.

```
pinMode(8, OUTPUT);       // il pin 8 manda segnali
pinMode(2, INPUT);         // il pin 2 legge segnali
pinMode(3, INPUT_PULLUP); // ingresso con resistenza interna
```

digitalWrite() Permette di inviare un valore logico (ON/OFF).

```
digitalWrite(8, HIGH);    // accende
digitalWrite(8, LOW);     // spegne
```

digitalRead() Usato per leggere il valore di un pulsante, un sensore digitale, ecc.

```
int stato = digitalRead(2);  // restituisce 0 o 1
```

analogRead() – Leggere un valore analogico Disponibile sui pin contrassegnati con "A0, A1, …". Restituisce un numero da *0 a 1023*, proporzionale alla tensione misurata.

```
int valore = analogRead(A0);
```

analogWrite() – Uscita PWM Simula un valore analogico tramite un segnale digitale molto veloce. Usato per regolare la luminosità di un LED o la velocità di un motorino.

```
analogWrite(pin, valore);   // PWM 0–255 (solo pin con ~)
```

delay() – Attendere un intervallo di tempo

millis() – Timer non bloccante Restituisce i millisecondi trascorsi dall'accensione.

```
delay(ms);                  // pausa in millisecondi
millis();                   // tempo trascorso dall'accensione
```

Serial.begin(), Serial.print() – Comunicazione seriale Usate per inviare dati al computer, utilissime per il debugging

```
Serial.begin(9600);
Serial.println("Hello!");
```

Nota: 9600 è la velocità di trasmissione dei dati, chiamata *baud rate* (9600 bit al secondo).

Arduino e il PC devono usare la stessa velocità, altrimenti: sul monitor seriale appaiono simboli strani, i dati sembrano "rovinati", o la comunicazione non è affidabile.

Funzioni personalizzate

In Arduino, come in moltissimi linguaggi di programmazione, possiamo creare **funzioni nostre** per organizzare meglio il programma. Una funzione è un "mini-programma" che esegue un compito specifico: accendere un LED, fare un calcolo, leggere un sensore...

Usare funzioni personalizzate rende il codice: più **ordinato**, più **facile da leggere**, più **riutilizzabile**.

Una funzione ha tre parti:

1. **il tipo di valore che restituisce** (es. int, float, void...)
2. **il nome della funzione**
3. **gli eventuali parametri** tra parentesi

```
int somma(int a, int b) {
      return a + b;
}
```

Per usare la funzione:

```
int risultato = somma(3, 4);  // risultato = 7
```

Se una funzione **non deve restituire niente**, usiamo void.

```
void accendiLED(int pin) {
      digitalWrite(pin, HIGH);
}
```

E viene usata :

```
accendiLED(8);   // accende il LED sul pin 8
```

Strutture di controllo

Le strutture di controllo permettono al programma di **prendere decisioni** o **ripetere delle operazioni**. Sono fondamentali perché rendono Arduino capace di reagire a ciò che succede (un sensore, un pulsante…) o di eseguire istruzioni più volte.

Istruzione if – prendere decisioni L'istruzione if controlla se una condizione è vera oppure no. Se la condizione è vera, Arduino esegue il primo blocco. Se è falsa, esegue l'istruzione sotto else.

```
if (temperatura > 30) {
      digitalWrite(8, HIGH);    // accendo una ventola
} else {
      digitalWrite(8, LOW);     // la spengo
}
```

Ciclo for – ripetere un numero preciso di volte È utile quando sappiamo quante volte vogliamo ripetere qualcosa.

```
for (int i = 0; i < 10; i++) {
      Serial.println(i);
}
```

Ciclo while – ripetere finché una condizione è vera Utile quando non sappiamo quante ripetizioni serviranno. Il ciclo **termina solo quando la condizione diventa falsa**.

```
while (digitalRead(2) == HIGH) {
      // continua finché il pulsante è premuto
}
```

Esempi

Accendere e spegnere un LED

```
void setup() {
      pinMode(13, OUTPUT);    // imposta il pin 13 come uscita
}

void loop() {
      digitalWrite(13, HIGH); // accende il LED
      delay(1000);            // attende 1000 ms (1 secondo)
      digitalWrite(13, LOW);  // spegne il LED
      delay(1000);
}
```

Come evitare errori comuni

- chiudere sempre parentesi { }
- ogni istruzione termina con ;
- rispettare maiuscole e minuscole
- non usare pin analogici con digitalWrite se non sono configurati correttamente
- se il codice non compila, leggere SEMPRE il primo errore in alto.