CORSO DI ARDUINO

Arduino, la magia dei sogni che diventano realtà…

Libera traduzione del Language Reference da Arduino.cc con qualche spunto preso dal sito maffucci.it del professor Michele Maffucci che mi ha aiutato molto, con i suoi esempi molto semplici e chiari a sfruttare Arduino. Grazie anche a tutti quelli che sulla rete condividono esempi e soluzioni a cominciare dal forum ufficiale di Arduino.

Giuseppe Tamanini, gennaio 2019

Rilasciato con licenza CC BY-SA

CORSO DI ARDUINO

Arduino, la magia dei sogni che diventano realtà…

CLOSED SOURCE (Copyright) – OPEN SOURCE (Copyleft)

Mentre il Closed Source (Sorgente chiuso), seppur nato con la buona intenzione di proteggere le proprietà intellettuali (invenzioni) creando i brevetti nell'hardware e le licenze d'uso nel software, tenendo segreta una tecnologia non consente ad altri di poterla migliorare (impedire da un punto di vista dell'economia generale affidarsi al segreto non è efficiente per l'innovazione tecnologica inoltre arricchisce poche persone), l'Open Source (Sorgente aperto) permette la condivisione delle conoscenze e quindi dà la possibilità a tutti di usare, testare e migliorare qualsiasi cosa venga divulgata in questo modo senza impedirne però eventuali guadagni.

CHE COS'È ARDUINO?

Essenzialmente una scheda Arduino è un elaboratore che legge degli input e restituisce degli output.
Arduino nasce in Italia nel 2005 presso l'Ivrea Interaction Design.
Arduino è Open Source, cosa che ha permesso la libera divulgazione di informazioni riguardanti il progetto stesso dell'hardware, comprendente gli schemi, la lista dei materiali, il layout dei dati del circuito stampato, solitamente insieme al Software libero (FOSS) per far girare l'hardware.
Essendo Arduino è una piattaforma elettronica open source basata su hardware e software facili da usare negli ultimi undici anni migliaia di persone in tutto il mondo lo hanno scelto per realizzare i propri progetti.
Le schede elettroniche Arduino sono molto piccole ed economiche, ma estremamente versatili: per questo motivo ha trovato posto nella casa di hobbisti, negli studi di professionisti e in molte scuole, dove può diventare una grande risorsa didattica.

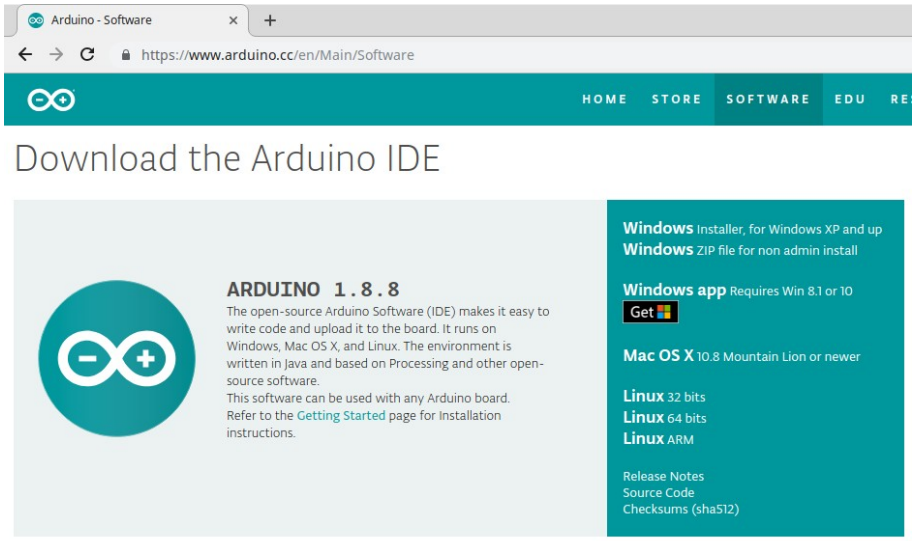
COSA SI PUÒ FARE CON ARDUINO?

Una risposta semplicistica potrebbe essere: fa qualsiasi cosa vogliate fargli fare.
Studenti e insegnanti lo usano per costruire strumenti scientifici economici, per verificare principi fisici e chimici, o per cominciare con la programmazione e la robotica. Progettisti e architetti costruiscono prototipi interattivi, musicisti e artisti lo usano per installazioni e per sperimentare con nuovi strumenti musicali. I maker, naturalmente, lo usano per realizzare molti dei progetti mostrati al Maker Faire di Roma. Arduino è uno strumento chiave per imparare nuove cose.
Si può usare per controllare le luci di casa, far funzionare un robot, lanciare un razzo amatoriale, controllare un diorama pieno di trenini, realizzare complessi strumenti scientifici e mille altre cose.; ognuno di noi può programmarlo usando l'Arduino Software (IDE), che è basato su un linguaggio semplificato che risponde al nome di Processing.

Arduino Uno rev.3: la versione più diffusa di Arduino al mondo. Presente in tantissimi "cloni" e varianti con nomi anche bizzarri: Funduino, Freeduino, GSTduino, Freaduino Diavolino, Fishino, ecc.

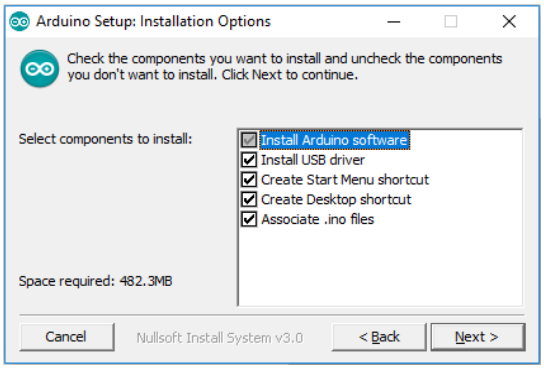
L'ARDUINO IDE

L'ambiente di sviluppo integrato (IDE) di Arduino è un'applicazione multipiattaforma in Java (disponibile quindi per Windows, Apple e Linux), ed è derivata dall'IDE creato per il linguaggio di programmazione Processing e per il progetto Wiring. È stata creata da Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis, per iniziare alla programmazione i neofiti, che siano a digiuno di pratica nello sviluppo di software.



Per permettere la stesura del codice sorgente, l'IDE include un editore di testo dotato inoltre di alcune particolarità, come il syntax highlighting, il controllo delle parentesi e l'indentazione automatica. L'editor è inoltre in grado di compilare e caricare sulla scheda Arduino il programma eseguibile in una sola passata e con un solo click.

Installazione:



Per installare l'IDE su Windows scaricare dal sito www.arduino.cc, dalla sezione download, il software arduino-1.8.9-windows.exe (o versione successiva) e avviarlo.
Dopo la richiesta di accettazione della licenza appare una finestra. Lasciate tutto come viene proposto e fate clic su Next.
Al termine troverete sul Desktopo l'icona dell'IDE di Arduino pronta per essere avviata.

Su Ubuntu Linux o Linux Mint, invece, conviene usare Ubuntu Make: strumento che ti consente di installare l'ultima versione dell'IDE di Arduino su Ubuntu Linux da linea di comando.

```
sudo apt-get install ubuntu-make
```

Se non usi Ubuntu 18.04+ o desideri installare le ultime versioni devi aggiungere il repository:

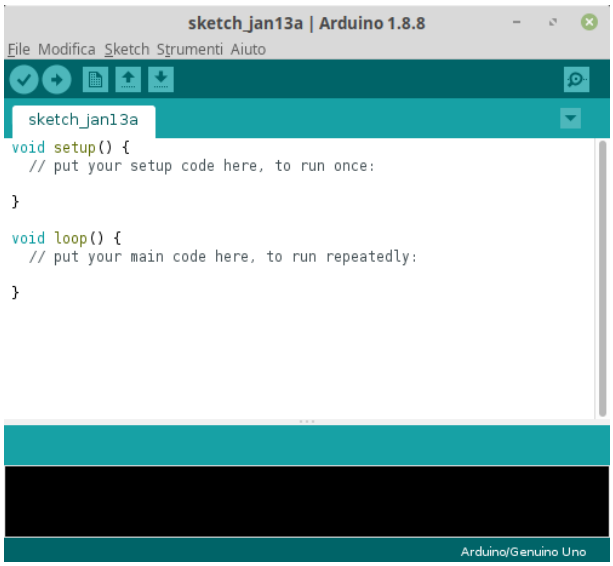
```
sudo add-apt-repository ppa:ubuntu-desktop/ubuntu-make
sudo apt-get update
```

Installa Arduino IDE con umake

```
umake ide arduino
```

Per poter usare Arduino però sarà necessario anche installare il driver apposito. Su Windows viene installato automaticamente quando viene collegato Arduino alla porta USB, su Linux verrà chiesto di installarlo al primo avvio dell'IDE.

L'INTERFACCIA



STRUTTURA DI UN PROGRAMMA

La struttura base del linguaggio di programmazione di Arduino si sviluppa sulla dichiarazione e impostazione delle librerie usate e dichiarazione delle variabili globali, seguite da due funzioni: void setup() e void loop().
Queste due funzioni racchiudono le necessarie istruzioni, sia per gestire i dispositivi collegati con Arduino che per svolgere il programma richiesto.

Esempio:

```
int ledpin = 3; // assegna a ledpin il valore 3
```

void setup( )

La funzione setup( ) è la prima ad essere chiamata quando parte uno sketch.
Viene utilizzata per inizializzare variabili, per impostare lo stato dei pin, per far partire le librerie da usare, per l'impostazione delle comunicazioni seriali.
La funzione di setup() sarà la prima ad essere eseguita dopo ogni accensione o reset di Arduino.

Sintassi:

```
void setup( ){
   // istruzioni da eseguire all'avvio;
}
```

Esempio:

```
void setup( ){
  Serial.begin(9600); // inizializza la seriale con velocità 9600 baud (bit per secondo)
  pinMode(ledpin, OUTPUT); // imposta il pin 3 (valore di ledpin) come uscita
}
```

void loop( )

Dopo la creazione della funzione setup(), che inizializza e imposta i valori iniziali, la funzione loop() fa proprio quanto suggerisce il proprio nome (loop significa ciclo da eseguire ripetutamente) eseguendo all'infinito il programma definito al suo interno.
Esegue il programma interagendo con la scheda Arduino.

Sintassi:
```
void loop( ){
  // istruzioni da ripetere in modo ricorsivo;
}
```

Esempio:

```
void loop( ){
  digitalWrite(ledpin, HIGH); // metti il pin 3 (valore di ledpin) allo stato alto
  delay(1000); // attendi 1 secondo
  digitalWrite(ledpin, LOW); // metti il pin 3 allo stato basso
  delay(1000); // attendi 1 secondo
}
```

SINTASSI
Alla fine di un'istruzione e per separare gli elementi del programma deve essere digitato un punto e virgola (;). Dimenticarlo produrrà errore di compilazione.

Esempio:

```
int valore = 13; // ; al termine di un'istruzione
for (int i = 0; i < 14; i++) { // ; come separatore degli elementi di un ciclo for
  // serie di istruzioni
}
```
Le parentesi graffe **{ }** definiscono l'inizio e la fine di un blocco di istruzioni e funzioni.
In un blocco si posso dichiarare e impiegare variabili locali che cessano di esistere al termine dell'esecuzione del blocco stesso.

Esempio:

```
void loop() {
  // istruzioni da ripetere all'infinito (fin quando l'Arduino non viene spento o resettato)
}
```

Le parentesi tonde **( )** si usano nelle strutture di controllo per racchiudere valori, condizioni e altri parametri.

Esempio:
```
  if (val == HIGH) { // se l'input è HIGH (pulsante premuto con 5 V sul pin 8)
    digitalWrite(LED, HIGH); //imposta il valore dell'output a HIGH (5 V) accendendo il led
  } else {
    digitalWrite(LED, LOW); //imposta il valore dell'output a LOW (0 V) spegnendo il led
  }
```

Le **parentesi quadre** [ ] si usano per creare un array (vengono anche chiamate matrici o variabili indicizzate) oppure per richiamare uno dei valori della matrice.

Esempio:

```
int numeri[5];
Serial.println(numeri[1]);
```

Ad una parentesi che si apre deve corrisponderne una che chiude che, quindi, devo essere di numero pari. Un numero dispari di parentesi causerà un errore di compilazione.

L'ambiente di sviluppo della programmazione di Arduino aiuta a capire se le parentesi sono state aggiunte correttamente. Se si seleziona una parentesi o si clicca sullo spazio che segue una parentesi, l'ambiente di sviluppo individua la parentesi abbinata evidenziandola con un rettangolo.

```
void setup(){
  Serial.begin(9600); // inizializza la seriale con velocità 9600 baud
  pinMode(pulsante, OUTPUT); // imposta il pin (3) come uscita
}
```

COMMENTI

I commenti, ignorati dal compilatore, vengono utilizzati dal programmatore per descrivere il codice.

Blocco di commenti /* .... */

Un blocco di commenti inizia con **/*** e termina con i simboli ***/**

Esempio:

```
/*Esempio 04: questo esempio permette di accendere
tre led di colore diverso a seconda di quante volte
viene premuto un pulsante*/
```

Linea di commento //

Quando si vuole commentare una linea o parte di essa basta precedere il commento da //

Esempio:

```
// legge lo stato del pulsante
```

Variabili
Le variabili sono dei cassetti nella memoria a cui vengono assegnati dei nomi e dei contenuti. Se non viene assegnato nessun valore la variabile conterrà, 0 (zero, false) o *vuoto*. I numeri decimali vanno scritti nel formato inglese con il punto. Il nome può contenere numeri, lettere e (_).

boolean
Può avere come valori solo: *false*, *true*
Mentre il valore false corrisponde al valore zero, il valore true corrisponde a qualsiasi valore diverso da zero compresi i valori negativi.

```
boolean acceso = false; // dichiara "acceso" di tipo boolean e gli assegna il valore false o 0
```

char
Viene utilizzato per memorizzare un carattere attraverso il suo codice ASCII e occupa un byte di memoria e ; va da -128 a 127 ed i caratteri vengono memorizzati in forma numerica tramite il loro codice ASCII. Nelle dichiarazioni i caratteri vanno scritti tra apici ('A').

```
char mioCarattere = 'A';  // dichiara "miocarattere" come char gli assegna la lettera A
char mioCarattere = 65;   // ha lo stesso risultato della dichiarazione precedente
```

byte
Memorizza numeri a 8 bit (1 byte) interi (senza decimali) ed hanno un range da 0 a 255.

```
byte numero = 180; // dichiara "numero" di tipo byte e gli assegna il valore 180
```

int
Gli interi sono dei tipi di dato usati per memorizzare numeri senza decimali e memorizzano valori a 16 bit (2 byte) nel range da -32.768 a 32.767.

```
int numero = 1500; // dichiara "numero" di tipo intero e gli assegna il valore 1500
```

unsigned int
A differenza di **int** memorizza solo numeri positivi e quindi può contenere valori da 0 a 65.535.

```
unsigned int numero = 50000; // dichiara "numero" di tipo unsigned int con valore 50000
```

long
Estende la dimensione degli interi, senza virgola, memorizzati con 32 bit (4 byte) e quindi il range dei valori possibili va da -2.147.483.648 a 2.147.483.647.

```
long numero = 90000; // dichiara "numero" di tipo long e gli assegna il valore 90000
```

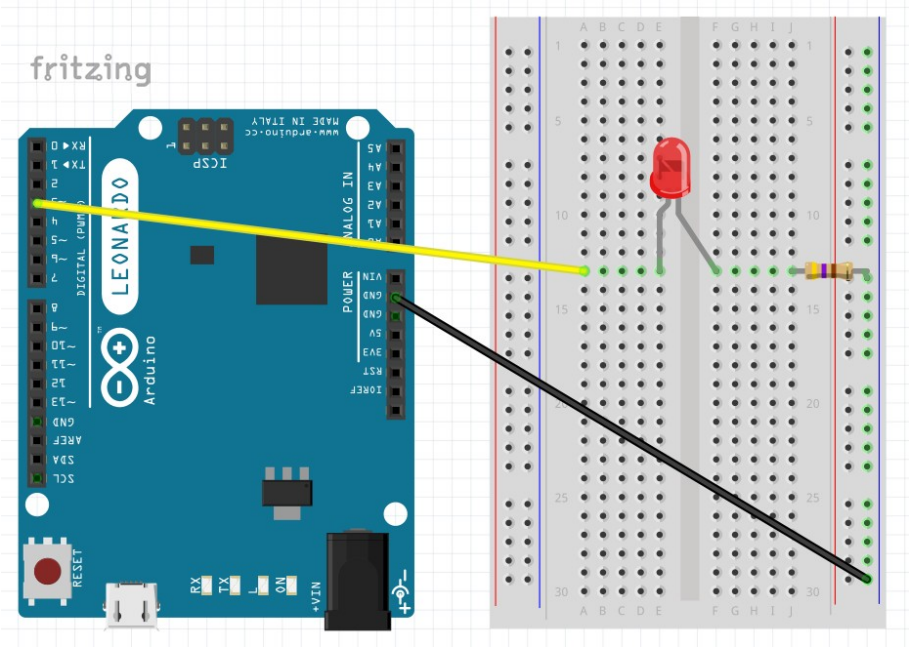
unsigned long
A differenza di **long** memorizza solo numeri positivi e quindi può contenere valori da 0 a 4.294.967.295.

```
unsigned long time;; // dichiara "time" di tipo unsigned long con valore 0
```

float
È un tipo di dato usato per i numeri in virgola mobile ed è usato per la contenere numeri piccolissimi o grandissimi con o senza segno e con o senza decimali. Utilizzano 32 bit (4 byte) nel range tra 3,4028235E+38 a -3,4028235E+38.

```
float numero = 3.14; // dichiara "numero" di tipo virgola mobile e gli assegna il valore 3,14
```

Esempio01: Facciamo lampeggiare un led collegato al pin 3



Ci servirà, oltre ad Arduino con il suo cavo USB, una breadboard, 1 led, una resistenza da almeno 100 ohm (io ho usato una da 470) e due ponticelli: uno nero (che consiglio di usare esclusivamente per la massa (GND)) e uno di un altro colore che non sia rosso (che consiglio di usare esclusivamente per le alimentazioni (+3,3V o 5V)).
Il programma è la somma degli esempi scritti nelle pagine precedenti.

Ora, dal menù **Strumenti**, selezionate, su **Scheda**, il tipo di Arduino che utilizziamo (Arduino Leonardo) e, su **Porta**, la porta a cui è collegato /dev/ttyACM0 (Arduino Leonardo). Su Windows la porta avrà nome COM seguito da un numero (Es. COM6).

Il pulsante **Verifica** permette di controllare se il nostro sketch è scritto correttamente. Se otteniamo il messaggio "Compilazione completata" significa che il programma è pronto per essere inviato al nostro Arduino. Altrimenti verranno segnalate le righe contenenti gli errori. Leggete attentamente i messaggi e correggete il codice.

Per caricare il programma su Arduino basta fare clic sul pulsante **Carica**, ma si può fare anche dal menù **Sketch** usando il comando **Carica** oppure premendo **CTRL + U**.

Se non avete salvato il programma vi verrà chiesto di farlo: “Salva la cartella dello sketch come...”. Scrivete, nella casella **Nome**, esempio01 e fate cli su Salva.
Al termine del caricamento verrà scritto “Caricamento Completato”. Il led lampeggerà lentamente.
Potete provare a modificare i due valori in delay() per cambiare i tempi di accensione e spegnimento. Tenete presente che il valore indica i millesimo di secondo: quindi se mettete valori inferiori a 100 vedrete il led, seppur meno luminoso, sempre acceso. Se i valori sono uguali il lampeggio sarà regolare (Es. 250, 250), se invece sono diversi il lampeggio sarà “zoppicante” (Es 250, 500).

Costanti
const oppure **#define** sono le parole chiave per creare una costante. È una una variabile “di sola lettura”. Ciò significa che la variabile può essere utilizzata come qualsiasi altra variabile del suo tipo, ma il suo valore non può essere modificato. Se si tenta di assegnare un valore a una costante si otterrà un errore del compilatore.
Le costanti rispettano le regole delle variabili.

Esempio:

```
const float pi = 3.14; oppure #define float pi = 3.14;
float x;

/...

x = pi * 2;    // questo è un esempio corretto dell’uso di pi

pi = 7;        // causerà errore: non è possibile sovrascrivere una costante
```

Note e avvertenze
Per la creazione di costanti numeriche o stringa èpossibile utilizzare sia **const** che **#define**. Per gli *array*, invece, si deve usare **const**.

Struttura di controllo (if)
Verifica una condizione ed esegue l'insieme di istruzioni se la condizione è "vera".

Sintassi:

```
if (condizione) {
  // istruzioni
}
```

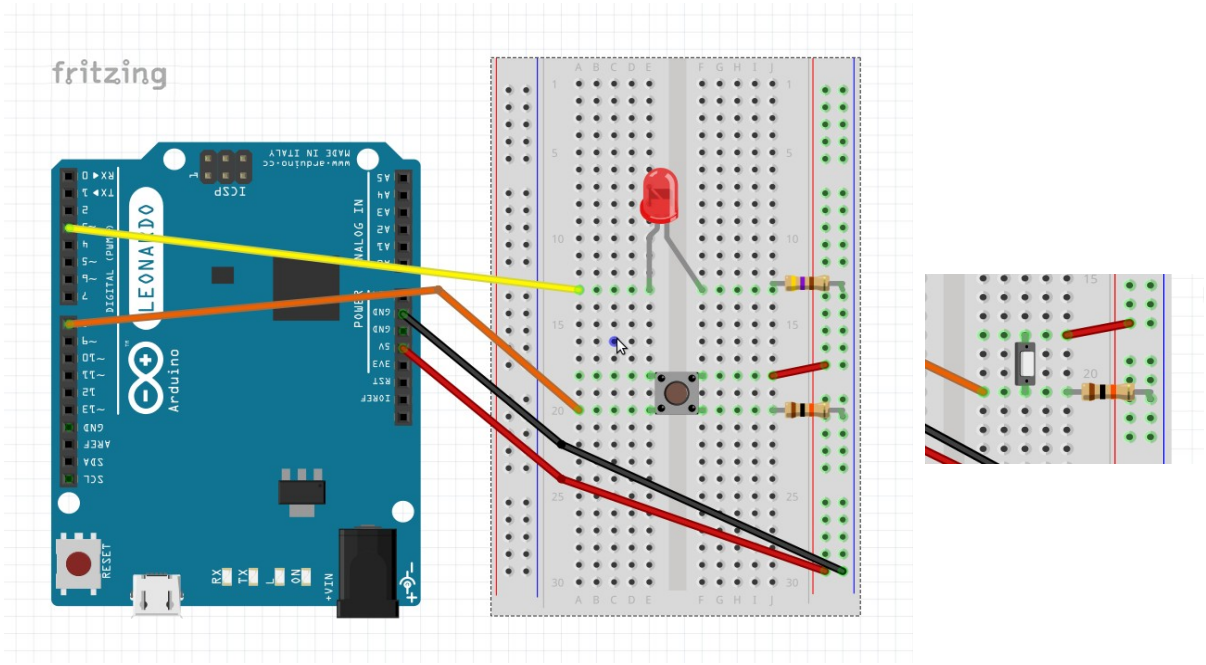
Struttura di controllo (if … else)
Verifica una condizione e, se la condizione è "vera", esegue un blocco di istruzioni. Altrimenti (else) se la condizione è “falsa” ne esegue un altro.

```
if (condition) {
  // istruzioni se vera
} else {
  // istruzioni se falsa
}
```

Esempio:

```
if (val == HIGH) { // se la variabile val è TRUE
  digitalWrite(LED, HIGH); // accende il led
} else { // in caso contrario
  digitalWrite(LED, LOW); // spegne il led
}
```

Esempio02: Accendiamo un led collegato al pin 3 alla premuta di un pulsante collegato al pin 8



Lo schema è lo stesso di prima con l'aggiunta di un pulsante (io ho usato, per comodità, uno a 4 piedini perché collegati elettricamente a due a due, se usate un pulsante a 2 pin, il filo arancione andrà collegato a destra sulla stessa fila della resistenza da 10K), una resistenza da 10K ohm e quattro ponticelli: uno rosso (+5V), uno nero (massa) e due di altro colore.

```
// Esempio 02: accendere un led quando il pulsante è premuto

#define LED 3 // imposta il pin dove verrà collegato il led
#define BUTTON 8 // imposta il pin dove verrà collegato il pulsante
int  val = 0; // la variabile val memorizzerà lo stato del pulsante

void setup() {
  pinMode(LED, OUTPUT); // imposta il pin del led come output
  pinMode(BUTTON, INPUT); // imposta il pin del pulsante come input
}

void loop() {
  val = digitalRead(BUTTON); // legge il valore (digitale) dell'input e lo memorizza in var

  if (val == HIGH) { // se l'input è HIGH (pulsante premuto con 5 V sul pin 8)
    digitalWrite(LED, HIGH); //imposta il valore dell'output a HIGH (5 V) accendendo il led
  } else {
    digitalWrite(LED, LOW); //imposta il valore dell'output a LOW (0 V) spegnendo il led
  }
}
```

Esempio03: Accendiamo un led collegato al pin 3 alla premuta di un pulsante collegato al pin 8, premendo una seconda volta il led si spegne

Lo schema rimane identico al precedente.

```
// Esempio 03: accende il led appena è premuto il pulsante mantenendolo acceso quando si rilascia
// premendo una seconda volta il pulsante spegne il led

#define LED 3 // memorizza il pin dove è collegato il LED
#define BUTTON 8 // memorizza il pin dove è collegato il pulsante
int val = 0; // memorizza lo stato del pin di input
int vecchio_val = 0; // memorizza lo stato del pin di input al passo precedente
int stato = 0; // memorizza lo stato in cui del led, stato = 0 led spento, stato = 1 led acceso
```

```
void setup() {
  pinMode(LED, OUTPUT); // imposta il pin digitale come output
  pinMode(BUTTON, INPUT); // imposta il pin digitale come input
}

void loop() {
  val = digitalRead(BUTTON); // legge il valore dell'input e lo conserva

  // se viene premuto il pulsante e prima non lo era
  if ((val == HIGH && vecchio_val == LOW)){
    stato = 1 - stato; // inverte lo stato (se è 1 diventa 0, invece 0 diventa 1)
    delay(15); // attesa di 15 millisecondi per evitare eventuali rimbalzi
  }

  vecchio_val = val; // memorizza il valore precedente di val

  if (stato == 1) { // se stato vale 1
    digitalWrite(LED, HIGH); // accende il led
  }
  else { // altrimenti
    digitalWrite(LED, LOW); //spegne il led
  }
}
```

Struttura di controllo (switch...case)

Switch confronta il valore di una variabile con i valori specificati nelle istruzioni case. Quando viene trovata un'istruzione case il cui valore corrisponde a quello della variabile viene eseguito il codice. La parola chiave *break* provoca l'uscita dalla truttura *switch* e viene in genere utilizzata alla fine di ciascun *case*. Senza un'istruzione *break* verranno eseguiti i comandi successivi fino alla fine dell'istruzione switch.

Sintassi:

```
switch (var) {
  case valore1:
    // istruzioni 1
    break;
  case valore2:
    // istruzioni 2
    break;
  case valore3:
    // istruzioni 2
    break;
}
```

oppure

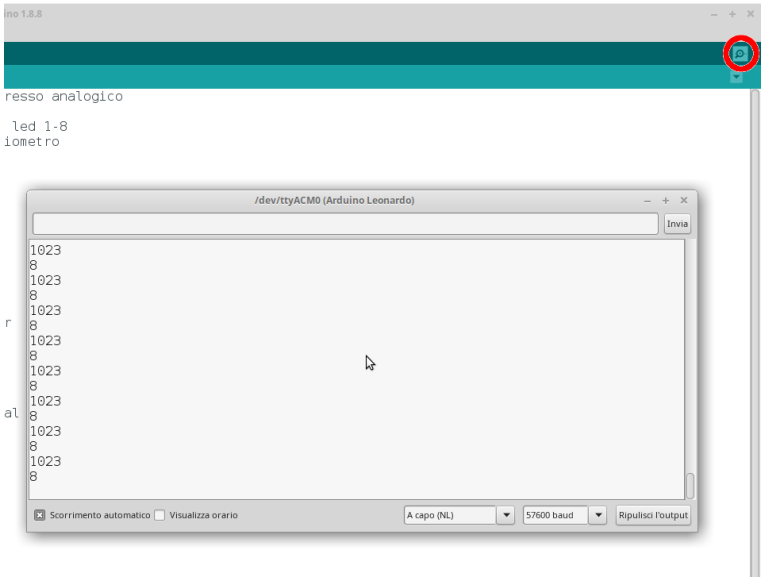
```
switch (var) {
  case valore1:
    // istruzioni 1
    break;
  case valore2:
    // istruzioni 2
    break;
  default:
    // istruzioni se nessun case precedente è stato eseguito
    break;
}
```

Parametri:
var: una variabile il cui valore è da confrontare nei vari case. Ammessi i tipi di variabili: int e char
valore1, *valore2*, ...: I valori di var che selezioneranno le istruzioni da eseguire.

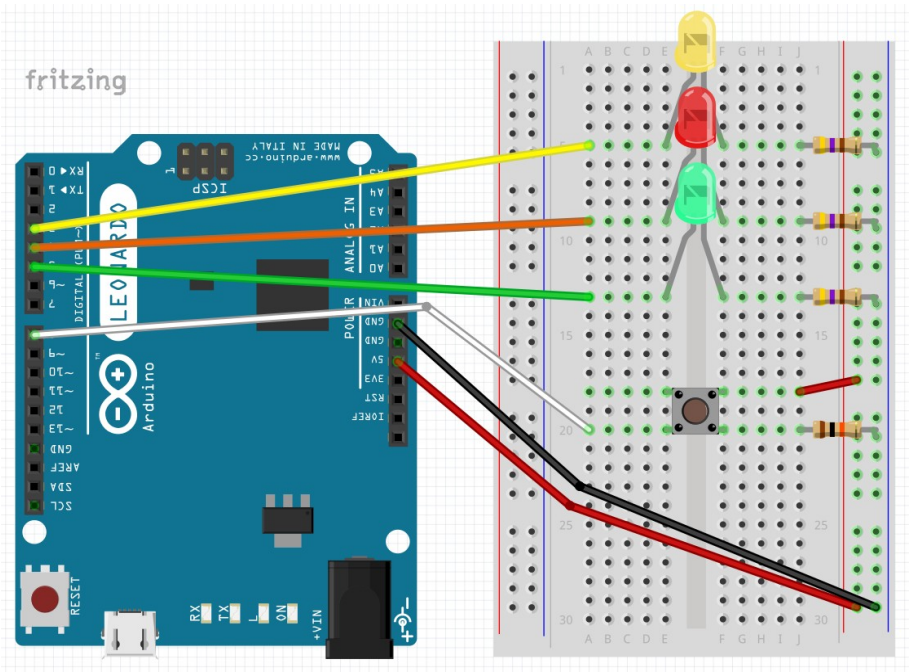
Il monitor seriale di Arduino IDE

Cliccando sul pulsante in alto a destra dell'IDE di Arduino si apre la finestra del monito seriale. In questo caso sta mostrando i dati dell'Esercizio 06: il valore di misurato all'ingresso del pin analogico e il numero dei led accesi.

Il monitor seriale è uno strumento utilissimo, quasi sempre l'unico, per poter testare il funzionamento del codice. È necessario selezionare il valore dei baud identico a quello definito in *Serial.begin()* nella funzione *void loop()*.



Esempio04: Accendiamo tre led di colore diverso collegati ai pin 3, 4 e 5, in base a quante volte viene premuto il pulsante collegato al pin 8



Lo schema è simile a quello precedente: con un led giallo, uno rosso e uno verde e rispettive resistenze da 470 ohm, un pulsante, una resistenza da 10K e sei ponticelli: uno rosso (+5V), uno nero (massa), e quattro di altro colore.

La resistenza da 10K di *pull-up* o di *pull-down* (come nell'esempio) serve per tenere l'ingresso di Arduino ad uno stato logico ben preciso quando il pulsante si trova col contatto aperto. Se non si portasse l'ingresso verso "massa" questo risulterebbe "flottante" e percepirebbe piccole tensioni che scambierebbe come se il pulsante fosse premuto, quindi si vedrebbe accendersi e spegnersi il led in maniera casuale. Non si può portare l'ingresso direttamente a massa, perché nel momento in cui si preme realmente il pulsante provocherebbe un corto circuito, per questo motivo si interpone una resistenza, di solito da 10k.

```
//Esempio 04: questo esempio permette di accendere
//tre led di colore diverso a seconda di quante volte
//viene premuto un pulsante

// costanti
#define ledYellowPin 3 // memorizza il pin dove verrà collegato il led giallo
#define ledRedPin 4 // memorizza il pin dove verrà collegato il led rosso
#define ledGreenPin 5 // memorizza il pin dove verrà collegato il led verde
#define buttonPin 8 //memorizza il pin dove verrà collegato il pulsante

// variabili
int buttonPushCounter = 0; // conta il numero delle volte che viene premuto il pulsante
int buttonState = 0; // memorizza lo stato corrente del pulsante
int lastButtonState = 0; //memorizza lo stato precedente del pulsante

void setup() {
  // setta l'uso del pin dove verrà collegato il pulsante come INPUT
  pinMode(buttonPin, INPUT);
  // setta l'uso del pin dove verranno collegati i led a OUTPUT
  pinMode(ledYellowPin, OUTPUT);
  pinMode(ledRedPin, OUTPUT);
  pinMode(ledGreenPin, OUTPUT);
  //inizializza la seriale e imposta la velocità di comunicazione a 9600 baud
  Serial.begin(9600);
}

void loop() {
  //legge lo stato del pulsante
  buttonState = digitalRead(buttonPin);
  delay(50); // ferma il programma per 50 millisecondi impedendo problemi di rimbalzo

  //se è cambiato lo stato del pulsante
  if (buttonState != lastButtonState) {
    //se lo stato corrente del pulsante è HIGH (premuto)
    if (buttonState == HIGH) {
      //se il pulsante è stato premuto 4 volte riporta la variabile a zero
      if (buttonPushCounter == 4) {
        buttonPushCounter = 0;
      }
      //incrementa il numero di premute
      buttonPushCounter++;
      //invia le informazioni sulla seriale
      Serial.println("premuto");
      Serial.print("per ");
      Serial.print(buttonPushCounter);
      Serial.println(" numero di volte");
    }
    else {
      //se lo stato corrente del pulsante è LOW (rilasciato)
      //invia sulla seriale la scritta "rilasciato"
      Serial.println("rilasciato");
    }
  }
  //assegna alla variabile lastButtonState lo stato corrente del pulsante
  lastButtonState = buttonState;

  //inizia la struttura case legata al valore di buttonPushCounter
  switch(buttonPushCounter)
  {

  //se buttonPushCounter vale 1 accende il led Giallo
  case 1:
```

```
    digitalWrite(ledYellowPin, HIGH);
    break;

  //se buttonPushCounter vale 2 accende il led Rosso
  case 2:
    digitalWrite(ledRedPin, HIGH);
    break;

  //se buttonPushCounter vale 3 accende il led Verde
  case 3:
    digitalWrite(ledGreenPin, HIGH);
    break;

 //se buttonPushCounter vale 4 spegne tutti i led
  case 4:
    digitalWrite(ledYellowPin, LOW);
    digitalWrite(ledRedPin, LOW);
    digitalWrite(ledGreenPin, LOW);
    break;
  }
}
```

Struttura di controllo (for

L'istruzione **for** serve per ripetere, per un determinato numero di volte, una serie di istruzioni, la ripetizione avviene se risulta vera una certa condizione logica, nel caso risulti falsa si esce dal ciclo. Un contatore (variabile intera) viene solitamente utilizzato per incrementare/decrementare e terminare il ciclo.

Sintassi:

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

L'*inizializzazione* (Es. `int i = 0`) avviene per prima ed una sola volta. Ogni ciclo successivo, la *condizione* (Es. `i <= 8`) viene testata; se è *true*, il blocco di istruzioni e l' *incremento/decremento* (Es. `i++`) viene eseguito, quindi la condizione viene testata di nuovo. Quando la condizione diventa *false*, il ciclo termina.

Esempio:

```
// Accende gradualmente un LED usando un pin PWM
int PWMpin = 10; // LED con in serie una resistenza da 470 ohm collegato al pin 10

void setup()
{
  // nessuna istruzione di setup necessaria
}

void loop() {
   for (int i = 0; i <= 255; i++) { // il contatore i viene fatto variare da 0 a 255
      analogWrite(PWMpin, i); // il led viene acceso gradualmente in base al valore di i
      delay(10); // aspetta 10 millisecondi
   }
}
```

Operatori di confronto:

Simbolo	Significato	Utilizzo
`==`	*uguale a*	`i == 0`
`!=`	*diverso da*	`i != 0`
`<`	*minore*	`a < 8`
`>`	*maggiore*	`a > 0`
`<=`	*minore o uguale*	`a <= 7`
`>=`	*maggiore o uguale*	`a >= 0`

Operatori logici:

Simbolo	Significato	Utilizzo
`&&`	*AND logico*	`i >= 0 && i < 10`
`||`	*OR logico*	`i >= 0 || j != 0`

ARRAY

Una matrice è una variabile che può contenere molti valori a cui si accede con un numero di indice: possiamo considerarla come una cassettiera a cui viene assegnato il nome della variabile, l'indice o gli indici se è multidimensionale, sono le coordinate (riga e, se esiste la colonna) dei cassetti.

Creare (dichiarare) un array
Tutti i metodi seguenti sono modi validi per creare (dichiarare) un array (matrice).

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

È possibile dichiarare un array senza inizializzarlo come in **myInts**.
In **myPins** dichiariamo un *array* senza darle dimensione. Il compilatore conta gli elementi e crea una matrice della dimensione appropriata.
Infine è possibile inizializzare e dimensionare l'array, come in **mySensVals** senza riempirla.
Si noti, che quando si dichiara una matrice di tipo char, è necessario un elemento in più rispetto all'inizializzazione, per contenere il carattere vuoto richiesto "\0" che viene aggiunto alla fine.

Accedere a un array
Gli array (matrici) sono creati a partire dall'indice zero, ovvero, facendo riferimento all'inizializzazione dell'array sopra, il primo elemento dell'array si trova con indice 0, quindi:

`mySensVals[0] == 2, mySensVals[1] == 4,` e così via.

Significa anche che in un array con dieci elementi, l'indice nove è l'ultimo elemento. Quindi:

```
int myArray[10]={9, 3, 2, 4, 3, 2, 7, 8, 9, 11};
     // myArray[9]   // contiene 11
     // myArray[10]   non è valido e può contenere informazioni casuali
                     (oltre l'indice massimo)
```

Per questo motivo devi fare attenzione nell'accedere agli array. L'accesso oltre la fine di un array (usando un numero di indice maggiore della dimensione dell'array dichiarato) legge dalla memoria che è in uso per altri scopi e questo non farà molto se non fornire dati non validi. Scrivere su posizioni di memoria casuali è, invece, sicuramente una cattiva idea e spesso può portare a risultati infelici come crash o malfunzionamenti del programma. Questo può anche essere un bug difficile da rintracciare. Il compilatore C non esegue alcun controllo per verificare se l'accesso dell'array rientra nei limiti legali della dimensione dell'array dichiarata.

Per assegnare un valore a un array:

```
mySensVals[0] = 10;
```

Per recuperare un valore da un array:

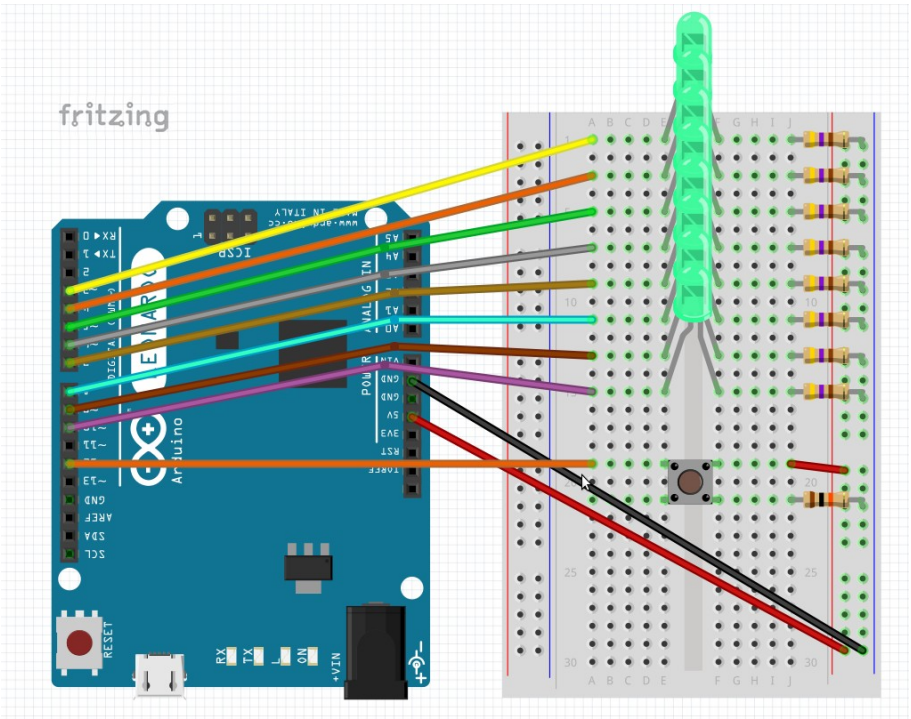
```
x = mySensVals[4];
```

Array e loop FOR

Gli array sono spesso usati all'interno della Struttura di controllo **For**, dove il contatore viene utilizzato come indice per ciascun elemento dell'array. Ad esempio, per stampare gli elementi di un array sulla porta seriale, potresti fare qualcosa del genere:

```
int i;

for (i = 0; i < 5; i = i + 1) {
  Serial.println(myPins[i]);
}
```

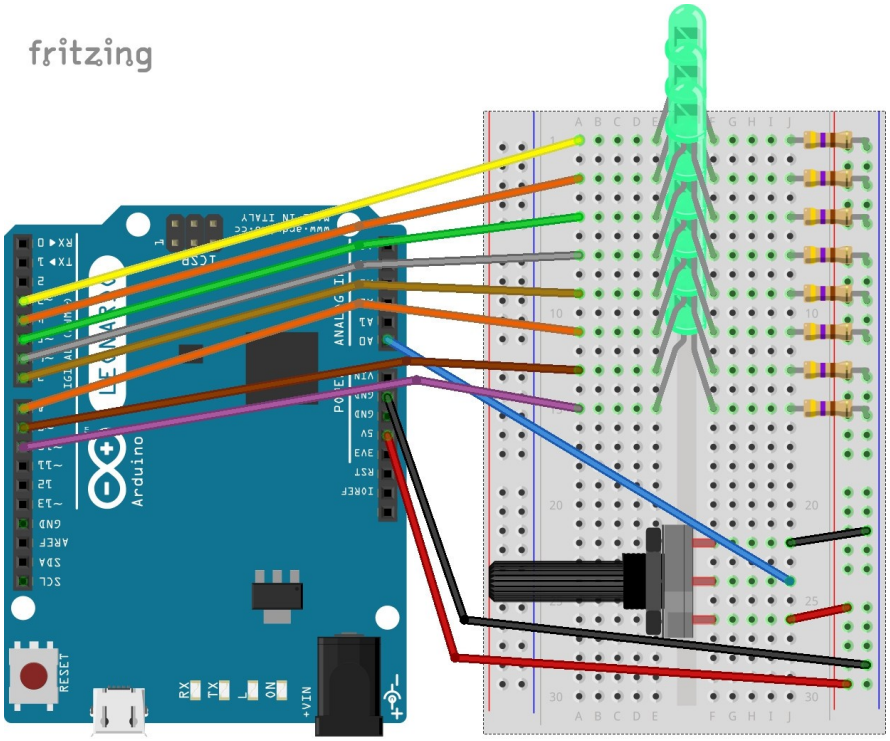
Esempio05: Accendiamo otto led dello stesso colore ai pin dal 3 al 10 in sequenza dall'alto verso il basso o viceversa in base a se sia premuto oppure no il pulsante.



Per realizzare questo esercizio servono 8 led (possibilmente dello stesso colore) e rispettive resistenze da 470 ohm, un pulsante, una resistenza da 10K e undici ponticelli: uno rosso (+5V), uno nero (massa), e nove di altro colore (anche lo stesso).

```
// Esempio 05: accendere 8 led in ordine se il pulsante è premuto, inverso quando non lo è

int LED[8] = {3, 4, 5, 6, 7, 8, 9, 10}; // imposta nell'array LED
                                         // il pin dove verrà collegato il led 1-8
#define BUTTON 12 // imposta il pin dove verrà collegato il pulsante

int  val = 0; // la variabile val memorizzerà lo stato del pulsante

void setup() {
  for (int i = 0; i < 8; i++) { // la variabile i varierà da 0 a 7
    pinMode(LED[i], OUTPUT); // imposta i pin da 3 a 10 dei led come output
  }
  pinMode(BUTTON, INPUT); // imposta il pin del pulsante come input
}

void loop() {
  val = digitalRead(BUTTON); // legge il valore (digitale) dell'input e lo memorizza in var
  int i; // definisce la variabile locale i come intero
  if (val == HIGH) { // se l'input è HIGH (pulsante premuto con 5 V sul pin 7)
    for (i = 0; i < 8; i++) { // la variabile i varierà da 0 a 7
      digitalWrite(LED[i], HIGH); // accendende il led LED con indice i
      delay(100); // aspetta 0,1 secondi
      digitalWrite(LED[i], LOW);  // spegne il led LED con indice i
    }
  } else {
    for (i = 7; i >= 0; i--) { // la variabile i varierà da 7 a 0
      digitalWrite(LED[i], HIGH); // accendende il led LED con indice i
      delay(100); // aspetta 0,1 secondi
      digitalWrite(LED[i], LOW);  // spegne il led LED con indice i
    }
  }
}
```

Esempio 06: accendere da 0 a 8 led in base al valore di tensione misurata da 0 a 5V su un ingresso analogico.



Per realizzare questo esercizio servono 8 led (possibilmente dello stesso colore) e rispettive resistenze da 470 ohm, un potenziometro da 10K lineare e tredici ponticelli: due rossi (+5V), due neri (massa), e nove di altro colore (anche lo stesso).

Il potenziometro funziona da partitore di tensione: ruotato al minimo sul piedino centrale ci saranno 0V (0 led accesi) e al massimo 5V (tutti gli 8 led accesi). A metà corsa 2,5 V e (i primi 4 led accesi).

```
// Esempio 06: accendere da 0 a 8 led in base al valore di tensione misurata
// da 0 a 5V su un ingresso analogico

int LED[8] = {3, 4, 5, 6, 7, 8, 9, 10}; // imposta nell'array LED il pin
                                         //dove verrà collegato il led 1-8
#define ING_A A0 // imposta il pin analogico dove è collegato
                 //  il piedino centrale del potenziometro
int sensorValue; // valore analogico letto sul pin A0
int level; // valore del numero dei led da accendere in proporzione al valore analogico letto

void setup() {
  Serial.begin(9600); // Inizializza la seriale a 9600 baud
  for (int i = 0; i < 8; i++) { // la variabile i varierà da 0 a 7
    pinMode(LED[i], OUTPUT); // imposta i pin da 3 a 10 dei led come output
  }
}

void loop() {
  sensorValue = analogRead(ING_A); // legge il valore (digitale) dell'input
                                   // e lo memorizza in  sensorValue
  Serial.println(sensorValue); // Invia il valore di sensorValue sulla seriale
  int i; // definisce la variabile locale i come intero
  level = sensorValue  * 8 / 1023; // calcola il numero dei led 0-
                                   // da accendere in proporzione al valore 0-1024
  Serial.println(level);  // Invia il valore level sulla seriale
  for (i = 0; i < level; i++) { // la variabile i varierà da 0 a level - 1
    digitalWrite(LED[i], HIGH); // accende il led LED con indice i
  }
  for (i = 7; i >= level; i--) { // la variabile i varierà da 7 a level
    digitalWrite(LED[i], LOW); // spegne il led LED con indice i
  }
  delay(250);  // aspetta 250 millisecondi
}
```

La formula

```
level = sensorValue  * 8 / 1023;
```

se scritta in questo modo:

```
level = sensorValue / 1023 * 8;
```

perché essendo un operazione con numeri interi e svolgendo per prima la divisione: qualsiasi valore, a parte 1023/1023 che da 1, da 0/1023 a 1022/1023 darà sempre 0.
Come me ne sono accorto? Direte voi! Con il monitor seriale:

```
Serial.println(level);
```

MULTITASKING

millis() vs delay()

Il comando delay() ferma l'esecuzione del programma per un certo tempo. Quando è in esecuzione questo comando, però, Arduino non rileva alcun input così come non invia alcun output, di conseguenza, se nel frattempo succede qualcosa, i cambiamenti non potranno essere rilevati.
L'uso di dela00y() è la scelta più facile in un programma dove gli eventi sono sequenziali e non è necessario rilevare o inviare dati in tempo reale.
Quando invece delay() non è assolutamente da usare?
Ad esempio se devo rilevare la durata della premuta di un pulsante. Se uso delay() per qualsiasi tempo di attesa superiore a un quarto di secondo – delay(250) -, in questo lasso di tempo, è possibile rilasciare il pulsante e poi ripremerlo senza che il programma se ne accorga. L'unica condizione è quella che il pulsante sia di premuto alla nuova rilettura. Ad esempio se volessi attivare una certa opzione quando tengo premuto il pulsante per un secondo, usando delay(1000) ho lo stesso risultato anche premendo il pulsante una volta e poi di nuovo allo scadere del secondo.

Il comando millis() restituisce il numero di millisecondi passati dall'avvio del programma su Arduino. Per memorizzare il tempo di questo comando si usa una variabile unsigned long int che permette di arrivare a circa 50 giorni di funzionamento ininterrotto prima di azzerarsi.

Esempio07: Accendiamo un led collegato al pin 13 alla premuta per almeno un secondo di un pulsante collegato al pin 10, premendo una seconda volta, per la stessa durata, il led si spegne

```
// Esempio 07 (variante dell'esercizio 3 con l'uso di millis() al posto di delay()):
// quando viene premuto il pulsante per almeno un tempo prestabilito al suo rilascio
//  si accende il led, premendo di nuovo il pulsante lo spegne

#define led1Pin 13 // (costante) numero del pin dove verrà collegato il led
#define button1Pin 10 // (costante) numero del pin dove verrà collegato il pulsante
boolean valbutton1; // stato del pulsante
boolean oldvalbutton1; // vecchio stato del pulsante
boolean statoLed; // stato del led
static unsigned long myTime; // tempo di rilettura dello stato del pulsante
long premuto = 1000; // tempo (millisecondi) premuta del pulsante per cambiare lo stato del led

void setup() {
  pinMode(led1Pin, OUTPUT); // setta il pin del pulsante come OUTPUT
  pinMode(button1Pin, INPUT); // setta il pin del pulsante come INPUT
}

void loop() {
  valbutton1 = digitalRead(button1Pin); // memorizza lo stato del pulsante
  // se il pulsante è premuto ma non lo era prima
  if (valbutton1 && !oldvalbutton1) {
    myTime = millis(); // memorizza il tempo trascorso dall'avvio del programma
    oldvalbutton1 = true; // pone il vecchio stato del pulsante a vero
  // se il pulsante è stato rilasciato ma era premuto in precedenza
  } else if (!valbutton1 && oldvalbutton1) {
    oldvalbutton1 = false;  // pone il vecchio stato del pulsante a falso
  }
  //se il pulsante è premuto, era premuto in precedenza e è trascorso per almeno il tempo di premuto
  if (valbutton1 && oldvalbutton1 && millis() - myTime > premuto) {
    statoLed = !statoLed; // inverte il valore di state (stato del led)
    digitalWrite(led1Pin, statoLed); // accende o spegne il led in base a statoLed
    oldvalbutton1 = false;  // pone il vecchio stato del pulsante a falso
  }
}
```

Questo esempio però ha un difetto: se si tiene premuto il pulsante il led lampeggerà fino al suo rilascio.

Ho risolto il problema aggiungendo un'altra variabile che ho chiamato `cambiato` che serve a memorizzare se il pulsante ha cambiato lo stato del led.

```
// Esempio 07_1 (variante dell'esercizio 3 con l'uso di millis() al posto di delay()):
// quando viene premuto il pulsante per almeno un tempo prestabilito al suo rilascio
//  si accende il led, premendo di nuovo il pulsante lo spegne

#define led1Pin 13 // (costante) numero del pin dove verrà collegato il led
#define button1Pin 11 // (costante) numero del pin dove verrà collegato il pulsante
boolean valbutton1; // stato del pulsante
boolean oldvalbutton1; // vecchio stato del pulsante
boolean statoLed; // stato del led
boolean cambiato; // lo stato del led è cambiato
static unsigned long myTime; // tempo di rilettura dello stato del pulsante
long premuto = 1000; // tempo (millisecondi) premuta del pulsante per cambiare lo stato del led

void setup() {
  pinMode(led1Pin, OUTPUT); // setta il pin del pulsante come OUTPUT
  pinMode(button1Pin, INPUT); // setta il pin del pulsante come INPUT
}

void loop() {
  valbutton1 = digitalRead(button1Pin); // memorizza lo stato del pulsante
  // se il pulsante è premuto ma non lo era prima
  if (valbutton1 && !oldvalbutton1) {
    myTime = millis(); // memorizza il tempo trascorso dall'avvio del programma
    oldvalbutton1 = true; // pone il vecchio stato del pulsante a vero
  // se il pulsante è stato rilasciato ma era premuto in precedenza
  } else if (valbutton1 == false && oldvalbutton1) {
    oldvalbutton1 = false;  // pone il vecchio stato del pulsante a falso
    cambiato = false; // pone la variabile cambiato a falso
  }
   //se il pulsante è premuto, era premuto in precedenza dopo essere stato rilasciato (cambiato è
falso) e è stato premuto per almeno il tempo di premuto
  if (valbutton1 && oldvalbutton1 && cambiato == false && millis() - myTime > premuto) {
    statoLed = !statoLed; // inverte il valore di state (stato del led)
    digitalWrite(led1Pin, statoLed); // accende o spegne il led in base a statoLed
    cambiato = true;  // lo stato del led è cambiato
  }
}
```

LIBRERIE DI ARDUINO

Le librerie sono delle componenti di codice aggiuntivo che vengono richiamate all'inizio dei programmi e che servono per consentire ad Arduino di comunicare con schede aggiuntive, sensori, rilevatori, ecc. attraverso i pin di input e di output.
Una libreria, per Arduino, è un codice contenente delle funzioni che consentono a componenti aggiuntivi di connettersi / interfacciarsi con Arduino senza che l'utente debba scervellarsi a scrivere decine di righe di codice per il loro utilizzo. Il concetto è molto simile a quello che nell'informatica tradizionale viene chiamato driver: come quello per una stampante.

Per ogni componente aggiuntivo come: motori, display lcd, servo, ecc. serve una propria libreria che dev'essere implementata nel codice (sketch) per facilitarne l'utilizzo con la scheda Arduino.

Esistono diverse famiglie di librerie, vediamole nel dettaglio:

librerie di sistema: queste librerie sono presenti "nativamente" nella scheda e sono quelle che consentiranno alla scheda di comprendere lo sketch. Si tratta di una componenti di basso livello che di fatto fa parte del firmware. Se non esistessero dovremmo scrivere il codice in linguaggio macchina.

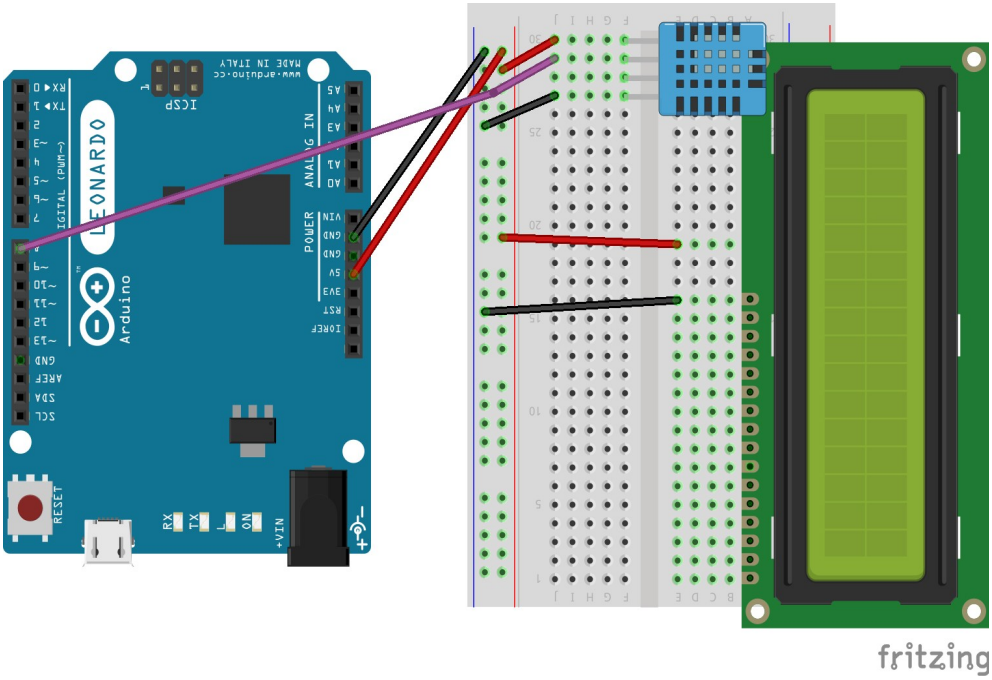
librerie esterne: queste librerie sono quello che appunto comunemente chiamiamo "librerie" e che servono all'IDE per comunicare con altri prodotti. Alcune sono già preinstallate insieme all'IDE di Arduino altre vanno installate successivamente: in modo automatico o in modo manuale. Per evitare problemi di conflitti con altre versioni è assolutamente consigliato l'uso automatico per l'installazione delle librerie.

La prima libreria che andremo ad utilizzare è quella del sensore di temperatura e umidità DHT11.

Per installare una libreria dall'IDE basta andare su **Sketch** > **#include libreria** > **Gestione librerie…**
Nella finestra che appare (impiegherà qualche secondo a caricare la lista completa), nella casella in alto a destra (Filtra la tua ricerca) scrivete dht11 e premete Invio.

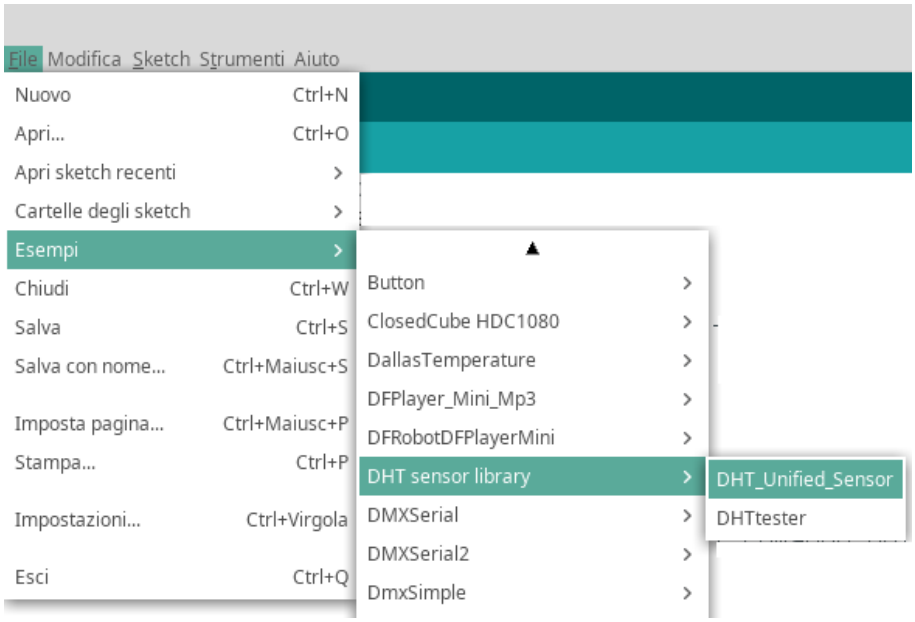
Fate clic sul pulsante Installa nella libreria *DHTT sensor library by Adafruit*.

Esempio08: Misuriamo temperatura e umidità ambientale con il sensore DHT11 con il pin DATA collegato al pin 2 dell'Arduino e visualizziamo i dati sul monitor seriale.



USIAMO L'ESEMPIO INTEGRATO NELLA LIBRERIE

Gli esempi si trovano nel menu File > Esempi.
Carichiamo quello che si chiama DHT_Unified_Sensor nel sotto menu DHT sensor library.

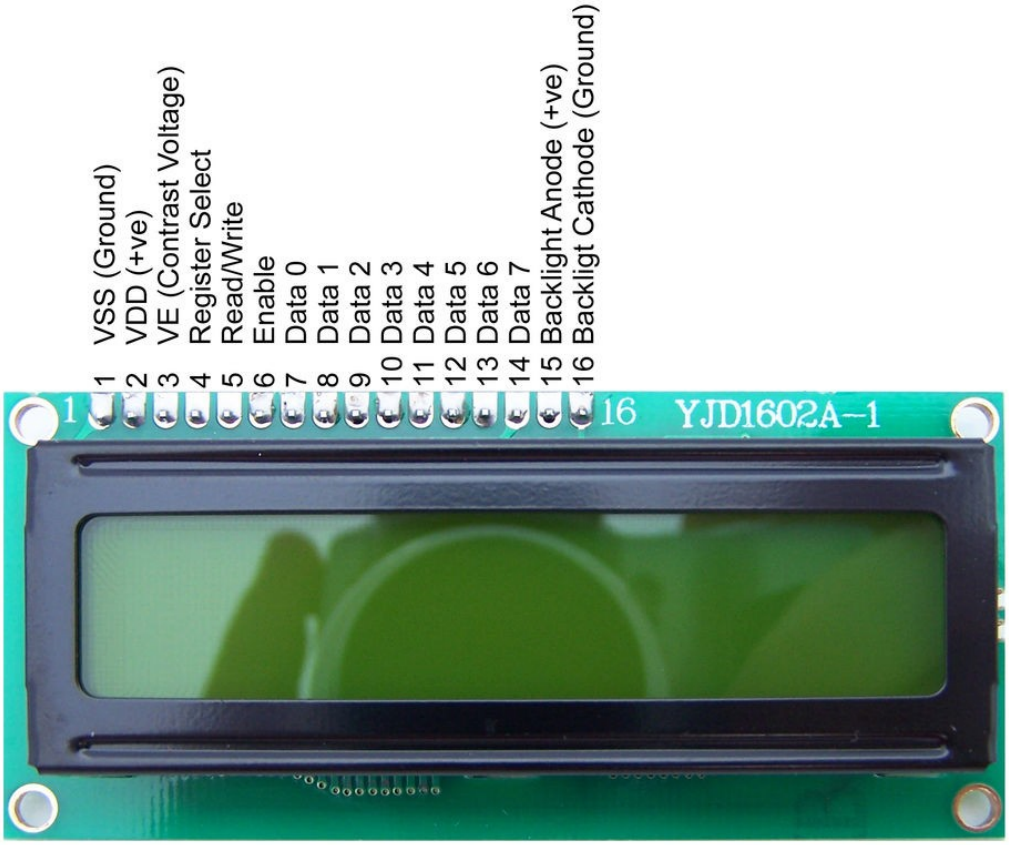


Essendo inutile inserire il programma esempio nel testo aggiungo solo il programma nella sua parte essenziale con i commenti tradotti in italiano. Ho tolto dalla struttura Setup() la parte che visualizza i dati di targa della sonda e dalla struttura Loop() i due if che visualizzano eventuali errori di lettura.

```
/*Esempio08: Misura temperatura e umidità ambientale
con il sensore DHT11 con il pin DATA collegato al pin 8
dell'Arduino e visualizziamo i dati sul monitor seriale.*/

// Librerie per l'uso del modulo RTC (data e l'ora)
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

#define DHTPIN 8 // pin di Arduino dove è collegato il pin DATA del DHT11
#define DHTTYPE DHT11 // indica alla libreria il tipo di sensore usato: DHT 11
DHT_Unified dht(DHTPIN, DHTTYPE);

uint32_t delayMS; // (unsigned int 32 bit (4 byte)) memorizza il tempo minimo di rilettura della
sonda in Microsecondi

void setup() {
  Serial.begin(9600); // inizializza la seriale
  // Inizializza il sensore DHT11
  dht.begin();
  sensor_t sensor;
  dht.temperature().getSensor(&sensor); // legge i dati di targa della sonda
  delayMS = sensor.min_delay / 1000; // memorizza il tempo minimo di rilettura della sonda
(/1000 per trasformare i microsecondi in millisecondi)
}

void loop() {
  delay(delayMS); // aspetta il tempo minimo di rilettura della sonda
  // Legge la temperatura e stampa il valore sulla seriale
  sensors_event_t event;
  dht.temperature().getEvent(&event);
  Serial.print("Temperatura: ");
  Serial.print(event.temperature);
  Serial.println("°C");
  // Legge l'umidità e stampa il valore sulla seriale
  dht.humidity().getEvent(&event);
  Serial.print("Umidità: ");
  Serial.print(event.relative_humidity);
  Serial.println("%");
```

Il Modulo LCD1602

Il display a cristalli liquidi LCD1602 o 1602 di tipo carattere, è un tipo di modulo a matrice di punti che serve per mostrare lettere, numeri e caratteri e così via. È composto da matrici di 5x7 punti; ogni matrice può mostrare un carattere. C'è un po' di spazio sia fra due caratteri (colonne) che tra le righe. Il modello 1602 significa che visualizza 2 righe di 16 caratteri (colonne).
Oltre ai caratteri ASCII standard, il pannello permette di creare dei caratteri personalizzati.
Questo modulo ha 16 pin di collegamento:



La libreria *LiquidCrystal* è già integrata nell'IDE di Arduino.
Per inserirla nel proprio programma basta aggiungerla da **Sketch** > **#include libreria** > **LiquidCrystal**

```
#include <LiquidCrystal.h>
```

Sempre nella parte iniziale del programma, delle dichiarazioni, la libreria *LiquidCrystal* richiede il comando LiquidCrystal lcd() che serve ad indicare alla libreria quali pin usare per comandare il pannello LCD.
LiquidCrystal lcd(*Register Select*, *Enable*, *Data 4*, *Data 5*, *Data 6*, *Data 7*);

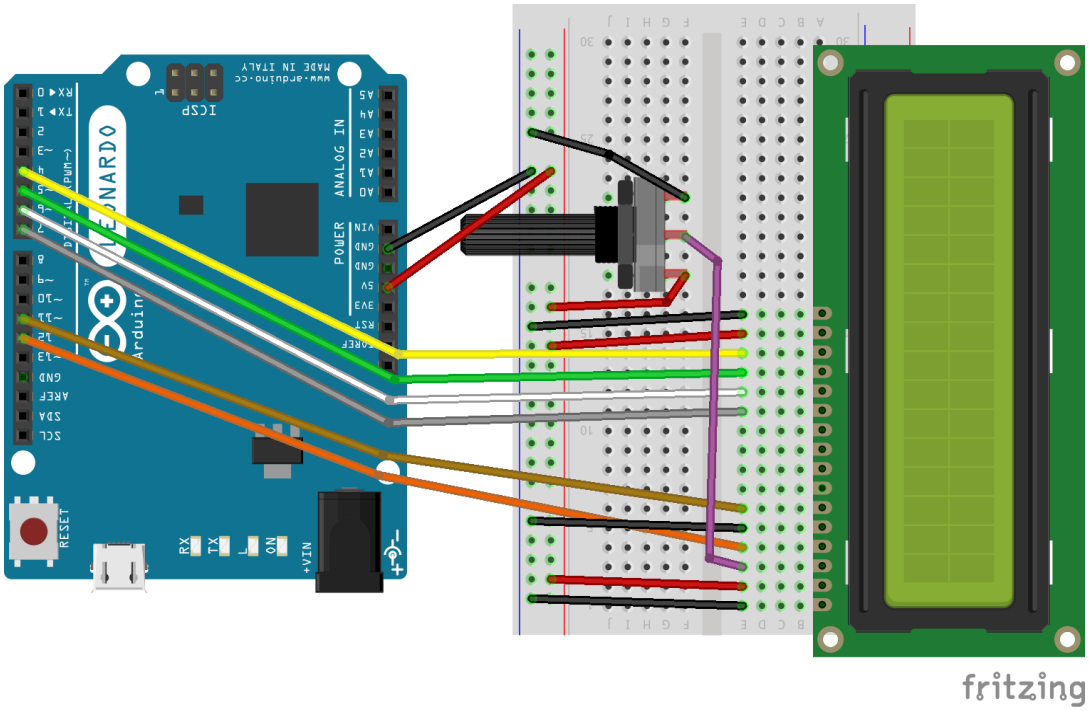
Con il comando:

```
LiquidCrystal lcd(12, 11, 7, 6, 5, 4);
```

I collegamenti tra LCD e Arduino sono:
* LCD RS (Register Select) pin to digital pin 12
* LCD Enable al pin 11
* LCD Data 4 al pin 7
* LCD Data 5 al pin 6
* LCD Data 6 al pin 5
* LCD Data 7 al pin 4

* LCD R/W (Read/Write) a massa (GND)
* LCD VSS a massa (GND)
* LCD VDD al 5V
* LCD Backlight Anode al +5V
* LCD Backlight Cathode a massa (GND)
* LCD VE (Contrast Voltage) al cursore del potenziometro da 10K.

Esercizio09: Visualizziamo la scritta Ciao mondo!” su un display LCD monocromatico da 16x2 caratteri.



Oltre al pannello LCD1602 servirà un potenziometro o un trimmer da 10K ohm che servirà per regolare la luminosità della scritta.

```
// Libreria per l'uso del pannello LCD
#include <LiquidCrystal.h>

// Indica alla libreria quali pin usare per il comando del pannello LCD
LiquidCrystal lcd(12, 11, 7, 6, 5, 4);

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

  // Avvia e setta il pannello LCD a 16 colonne e 2 riche
  lcd.begin(16, 2);
  lcd.print("Ciao mondo!"); // scrive sul pannello LCD
}

void loop () {
  // non sono necessari comandi
}
```

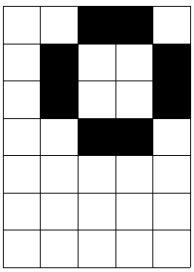
Ora aggiungiamo all’interno della struttura **loop()** il codice che serve a scrivere sulla seconda riga un numero crescente ogni 1000 millisecondi.

```
void loop () {
  for (int i = 0; i < 1000; i++) { // per i (intero) che varia da 0 a 999
    lcd.setCursor(0, 1); // posiziona l'inizio scrittura sul pannello alla colonna 0 (prima) e
riga 1 (prima)
    if (i < 10) { // se i minore di 10
      lcd.print("00"); // scrive sul pannello LCD due 00 davanti
    } else if (i < 100) { // altrimenti se i minore di 100
      lcd.print("0"); // scrive sul pannello LCD uno 0 davanti
    }
    lcd.print(i); // scrive il valore di i sul pannello LCD
    delay(1000); // aspetta 1 secondo
  }
}
```

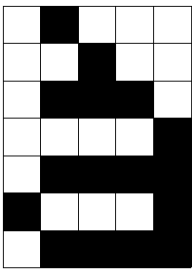
Creazione di caratteri personalizzati

Nel prossimo esercizio, per visualizzare la temperatura e l'umidità, avremo bisogno di visualizzare sul display due caratteri non standard: la a accentata (à) e il grado (°): caratteri che non sono fanno parte della tabella ASCII STANDARD.

Dobbiamo, in una matrice di 5x7, colorare i quadretti che vogliamo si accendino. Per creare il carattere dobbiamo dichiarare, nell'header del programma, una matrice di tipo byte da 8 valori (righe) contenenti ciascuno 5 valori binari (inserire una B all'inizio del numero binario) 0 indicherà che il pixel resterà spento e 1 acceso.



```
byte grado[8] = {
  B00110,
  B01001,
  B01001,
  B00110,
  B00000,
  B00000,
  B00000,
};
```



```
byte aaccentata[8] = {
  B01000,
  B00100,
  B01110,
  B00001,
  B01111,
  B10001,
  B01111,
};
```

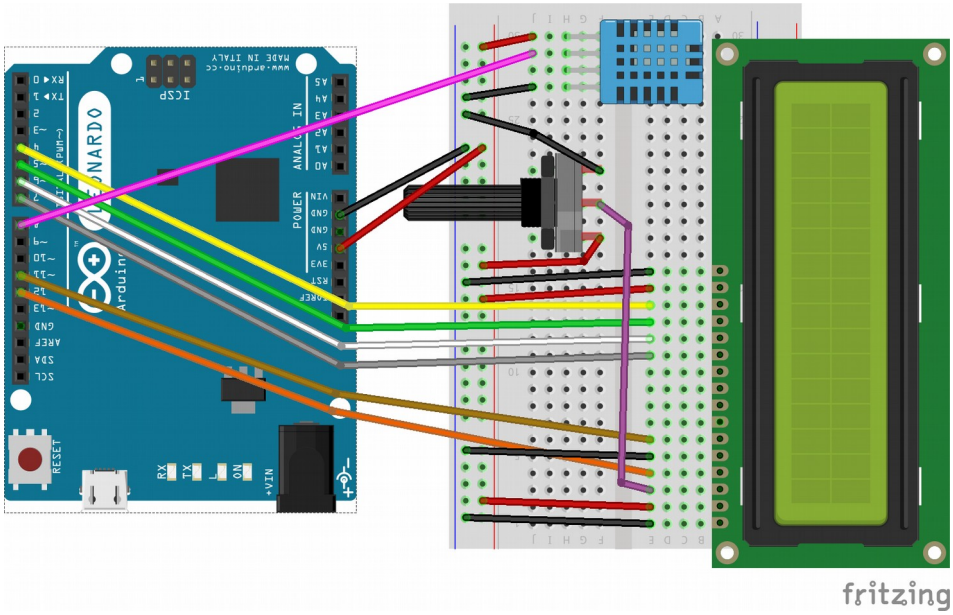
Per crearli, ne potremo utilizzare massimo 8 (da 0 a 7), in modo da poterli utilizzare nel nostro programma dobbiamo inserire nella procedura Setup() i comandi:

```
  lcd.createChar(0, grado);
  lcd.createChar(1, aaccentata);
```

Per stamparli nella prodedura Loop() dobbiamo usare i comandi:

```
lcd.write(byte(0)); // visualizza il carattere grado (°)
lcd.write(byte(1)); // visualizza la a accentata (à)
```

Esercizio10: Usando un pannello LCD1602 e un sensore di temperatura e umidità DHT11, visualizzare sul display la temperatura e l'umidità dell'ambiente.



Aggiungiamo allo schema precedente il sensore DHT11 collegando il pin .

```
/*Esercizio10: Usando un pannello LCD1602 e un sensore di temperatura e umidità DHT11,
visualizzare sul display la temperatura e l'umidità dell'ambiente.*/

// Librerie per l'uso del modulo HTC11
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

#define DHTPIN 8 // pin di Arduino dove è collegato il pin DATA del DHT11
#define DHTTYPE DHT11 // indica il tipo di sensore usato: in questo caso il DHT 11
DHT_Unified dht(DHTPIN, DHTTYPE);

// Libreria per l'uso del pannello LCD
#include <LiquidCrystal.h>

// Indica alla libreria quali pin usare per il comando del pannello LCD
LiquidCrystal lcd(12, 11, 7, 6, 5, 4);

uint32_t delayMS; // (unsigned int 32 bit (4 byte)) memorizza il tempo minimo di rilettura
della sonda in Microsecondi

// matrice che disegna il carattere grado
byte grado[8] = {
  B00110,
  B01001,
  B01001,
  B00110,
  B00000,
  B00000,
  B00000,
};

// matrice che disegna il carattere a accentata
byte aaccentata[8] = {
  B01000,
  B00100,
  B01110,
  B00001,
  B01111,
  B10001,
  B01111,
};
```

```
void setup () {
  Serial.begin(9600); // inizializza la seriale
  // Inizializza il sensore DHT11
  dht.begin();
  sensor_t sensor;
  dht.temperature().getSensor(&sensor);  // legge i dati di targa della sonda
  delayMS = sensor.min_delay / 1000; // memorizza il tempo minimo di rilettura della sonda
(/1000 per trasformare i microsecondi in millisecondi)

  // Avvia e setta il pannello LCD a 16 colonne e 2 riche
  lcd.begin(16, 2);
  lcd.createChar(0, grado); // crea il carattere grado
  lcd.createChar(1, aaccentata); // crea il carattere aaccentata
}

void loop () {
  delay(delayMS); // aspetta il tempo minimo di rilettura della sonda
  lcd.setCursor(0, 0); // posiziona l'inizio scrittura sul pannello alla colonna 0 (prima) e
riga 0 (prima)
  // Legge la temperatura e visualizza il valore sul display
  sensors_event_t event;
  dht.temperature().getEvent(&event);
  lcd.print("Temperatura:");
  lcd.print((int)event.temperature);
  lcd.write(byte(0)); // scrive il carattere grado
  lcd.print("C");
  lcd.setCursor(0, 1); // posiziona l'inizio scrittura sul pannello alla colonna 0 (prima) e
riga 1 (seconda)
  // Legge l'umidità e stampa il valore sulla seriale
  dht.humidity().getEvent(&event);
  lcd.print("Umidit");
  lcd.write(byte(1)); // scrive il carattere a accentata
  lcd.print(":");
  lcd.print((int)event.relative_humidity);
  lcd.print("%");
}
```

funzioni
Una funzione è un blocco di istruzioni racchiuso fra parentesi graffe {} a cui è attribuito un nome. Blocco di istruzioni che vengono eseguite quando nel programma viene digitato il nome della funzione.

Le funzioni possono essere usate per eseguire compiti ripetitivi e rendere più facilmente leggibile un programma.
Le funzioni possono, come per le variabili, essere di tipo: **char**, **int**, **double**, **float**, con l'aggiunta di **void** di tipo indefinito che non richiede parametri e non deve restituire alcun valore, come per void setup() e void loop().

Sintassi:
```
tipoFunzione nomeFunzione (parametri) {
  elenco istruzioni;
  ...
  ...
}
```

Esempi:
```
int somma(int a, int b) { // esempio 1
  int c = a + b;
  return c
}

void loop() {
  int c = somma(7, 5); // esegue la funzione somma() che calcola 7 + 5
                       // e restituisce il valore in c
  ...
}


... // esempio 2
void ledOn() {
  digitalWrite(pinLed,HIGH);
}

void ledOff() {
  digitalWrite(pinLed,LOW);
}

void loop() {
  ledOn(); // esegue la funzione leOn() che accende il led
  delay(1000);
  ledOff(); // esegue la funzione leOff() che spegne il led
  delay(1000);
}

... // esempio 3
void loop() {
  flashLed(13, 250); // esegue la funzione flashLed() che fa lampeggiare
                         // il led collegato al pin 13 a 250 ms
}

void flashLed(boolean ledPin, int flash) {
  digitalWrite(ledPin, HIGH);
  delay(flash);
  digitalWrite(ledPin, LOW);
  delay(flash);
}
```

Generare un numero casuale

Prima di generare un numero pseudo-casuali bisogna inizializzare il generatore di numeri pseudo-casuali con il comando **randomSeed()** specificando un valore come "seme".

```
randomSeed(numero) // numero per inizializzare la sequenza pseudo-casuale
```

Quando viene indicato lo stesso numero la sequenza all'avvio o al reset di Arduino sarà sempre la stessa. Per inizializzare il generatore di numeri casuali con un numero abbastanza casuale, basta leggere un pin analogico non connesso con analogRead().

Per generare il numero pseudo-casuale usare il comando **random()**. I parametri **min** e **max** sono di tipo **long**.

```
random(max) // il valore generato sarà compreso fra 0 e max - 1
random(min, max) il valore generato sarà compreso fra min e max - 1
```

Esempio:

```
void setup() {
  randomSeed(0) // la sequenza sarà sempre la stessa
}

void loop() {
  int a = random(300); // a sarà compreso fra 0 e 299
  Serial.println(a);
  int b = random(10, 20); // b sarà compreso fra fra 10 e 19
  Serial.println(b);
  delay(5000);
}
```

Il comando **return**
Serve per terminare una funzione e, se specificato, restituire un valore alla funzione di chiamata.

Sintassi:

```
return;

return variabile;
```

Esercizio 11: Genera due numeri pseudo casuali a e b e ne calcola la somma

```
void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(7)); // il seme è il valore letto sul pin analogico 7
}

void loop() {
  int a = random(0, 100); // a sarà compreso fra 0 e 99
  int b = random(0, 100);  // a sarà compreso fra 0 e 99
  int c = somma(a, b); // esegue la funzione somma() che restituisce il risultato in c
  Serial.print(a); // Visualizza i dati sulla seriale
  Serial.print(" + ");
  Serial.print(b);
  Serial.print(" = ");
  Serial.println(c);
  delay(5000); // aspetta 5 secondi
}

int somma(int a, int b) { // funzione somma
  int c = a + b; // mette il risultato di a + b in c
  return c; // restituisce c al programma chiamante
}
```

Il comando **return** può essere usato per escludere, per la localizzazione di eventuali errori, una parte di programma all'interno della funzione void loop():

Esempio:

```
void loop() {
  elenco istruzioni
  ...
  ...
  ...
  ...
  return;
  elenco istruzioni(che non verranno eseguite)
  ...
  ...
  ...
}
```

La comunicazione I²C

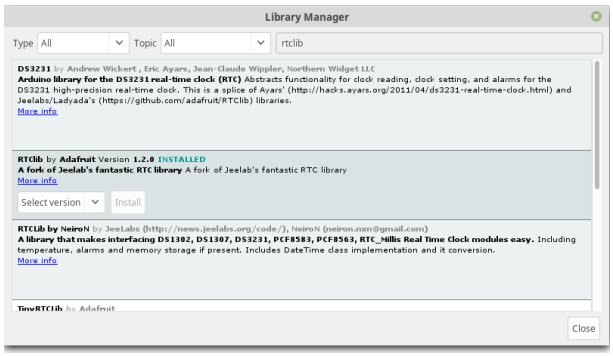
I²C (abbreviazione di Inter Integrated Circuit), (pronuncia i-quadro-ci o i-due-ci), è uno standard di comunicazione creato da Philips per permettere a più dispositivi di comunicare tramite un semplice bus composto da 2 fili.
Ogni dispositivo ha sul bus un indirizzo univoco (ID). e può quindi essere contattato in maniera semplice aprendo una connessione specificando il suo ID.
Ogni dispositivo può ricevere dati e spedire dati, a seconda dei comandi.
In Arduino per utilizzare questo tipo di comunicazione si usa la libreria wire e i pin identificati con SDA e SCL che però corrispondono ad altri pin come da tabella seguente. Solo Arduino Due ha una seconda interfaccia I2C (SDA1, SCL1) che non utilizza pin esistenti.

Scheda	*Pin I2C/TWI*
Uno, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1 , SCL1

Se si utilizza l'interfaccia I2C non si possono usare i pin A4, A5 su Arduino Unoù; 20, 21 su Arduino Mega e Due; 2, 3 su Leonardo.

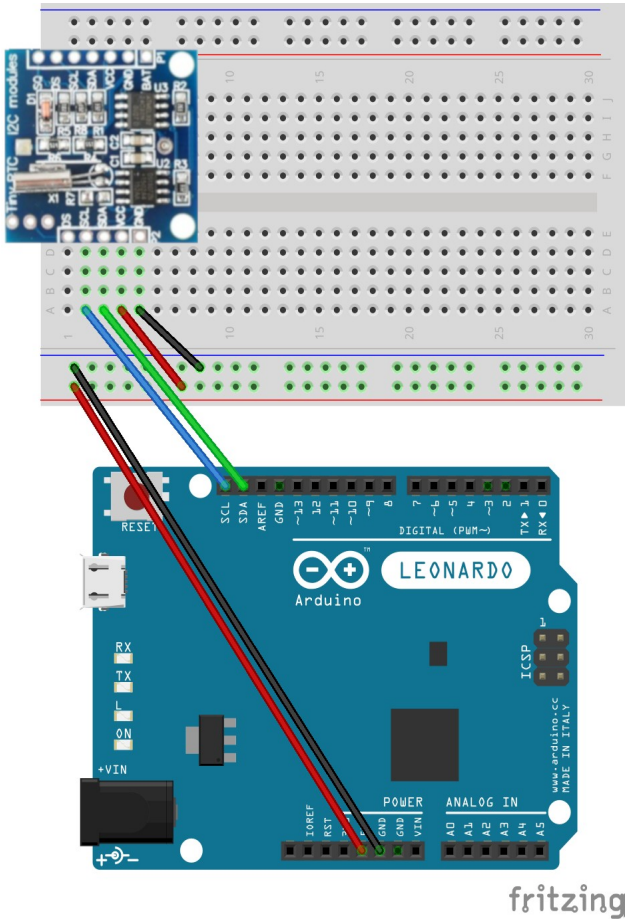
La libreria RTC

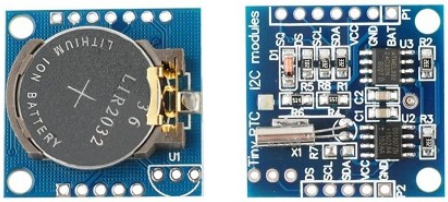
Arduino anche se ha al suo interno ha un orologio (tempo passato dall'accensione che però si azzera ogni volta che si spegne o si resetta o, comunque, dopo circa 50 giorni di funzionamento ininterrotto) non è adatto per avere data e ora correnti. Se vogliamo che Arduino conosca la data e l'ora esatta dobbiamo collegare ad esso un modulo esterno (Real Time Clock) dotato di una batteria tampone per evitare la perdita dei dati in caso di mancanza temporanea dell'alimentazione, connesso tramite l'interfaccia I2C di Arduino ai pin SDA e SCL.
La libreria che utilizzeremo sarà RTClib by Adafruit che, se non installata, potete installarla dal Library Manager, scrivento rtclib nella casella di ricerca.



Esercizio 10_1: Visualizzare sulla seriale la data e l'ora letta tramite collegamento I2C da un modulo RTC DS1307.

Schema di collegamento della scheda RTC DS1307 all'Arduino:





Il programma per visualizzare la data e l'ora sulla seriale:

```
// Librerie per usare DS1307 RTC connesso via I2C
#include <Wire.h>
#include "RTClib.h"

// inizializza l'interfaccia
RTC_DS1307 rtc;

// matrice che memorizza il nome dei giorni della settimana - dayOfTheWeek() = 0 è Domenica
char  daysOfTheWeek[7][9]  =  {"Domenica",  "Lunedì",  "Martedì",  "Mercoledì",
"Giovedì", "Venerdì", "Sabato"};

void setup() {
  Serial.begin(57600);
  // verifica la presenza del modulo RTC
  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  if (!rtc.isrunning()) { // se il modulo non è avviato
    Serial.println("RTC is NOT running!"); // scrive sulla seriale che il modulo
non è avviato
    // per settare la data e l'ora inserire i valori tra le parentesi
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // Questo è un esempio che setta la data al 12 aprile 2019 ore 2:28
   // i dati vanno inseriti secondo questo ordine: anno, mese, giorno, ora, minuti, secondi
```

```
    // rtc.adjust(DateTime(2019, 4, 12, 2, 28, 0));
  }
}

void loop() {
  DateTime now = rtc.now(); // legge i dati dal modulo RTC
  Serial.print(now.year(), DEC); // stampa l'anno
  Serial.print('/');
  Serial.print(now.month(), DEC); // stampa il mese
  Serial.print('/');
  Serial.print(now.day(), DEC); // stampa il giorno
  Serial.print(" (");
  // Visualizza i nomi dei giorni della settimana estraendo i valori dalla matrice
  Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
  Serial.print(") ");
  Serial.print(now.hour(), DEC); // stampa l'ora
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':'); // stampa i minuti
  Serial.print(now.second(), DEC);
  Serial.println(); // stampa i secondi
  delay(1000); aspetta 1 secondo
}
```