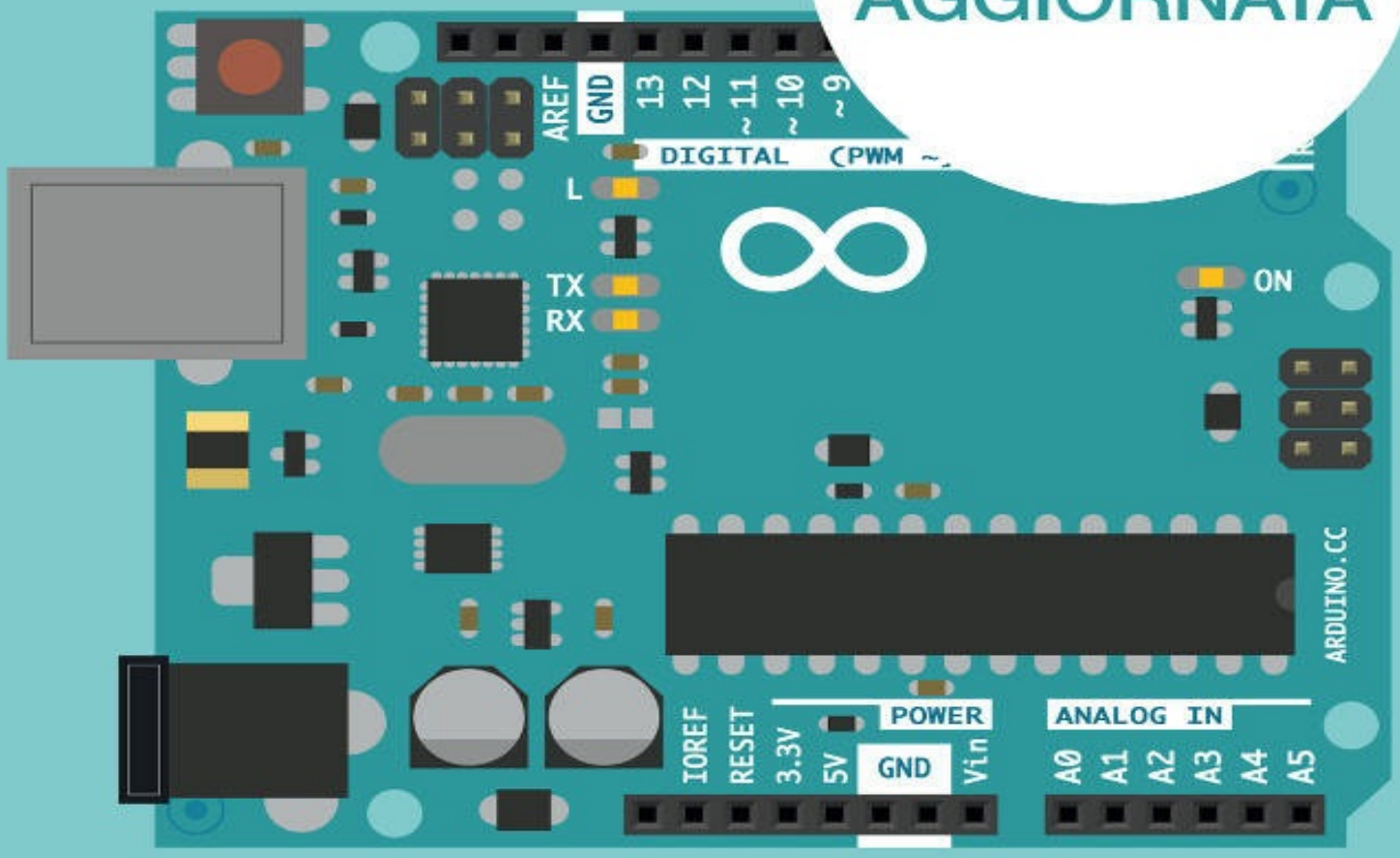
Piccolo manuale di Arduino

Il cuore della robotica fai da te

MATTEO TETTAMANZI

NUOVA EDIZIONE AGGIORNATA



APOGEO

POCKETCOLOR

Matteo Tettamanzi

ISBN: 9788850317356

Illustrazioni dei circuiti sulle breadboard grazie a Fritzing (http://fritzing.org).

L'edizione cartacea è in vendita nelle migliori librerie.

~

Sito web: www.apogeonline.com

Scopri le novità di Apogeo su Facebook

Seguici su Twitter @apogeonline

Rimani aggiornato iscrivendoti alla nostra newsletter

Il progetto Arduino è nato nel 2005 presso l'Interaction Design Institute di Ivrea, con l'obiettivo di fornire agli studenti un'alternativa semplice ed economica alle tradizionali piattaforme di prototipazione elettronica utilizzate per i progetti durante i corsi. I componenti del team che ha dato vita ad Arduino sono Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis.

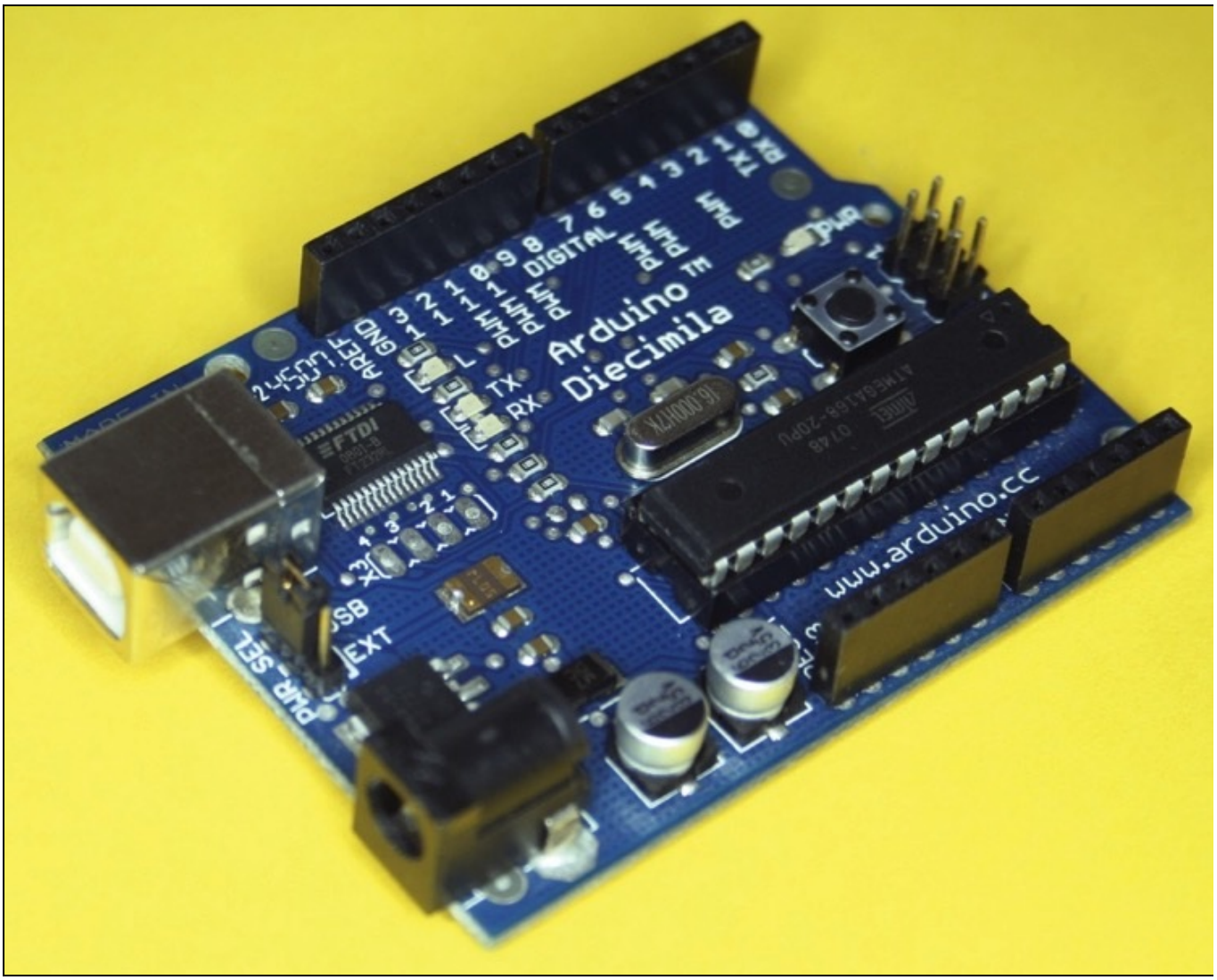


Figura I.1 Arduino Diecimila, una delle prime versioni della scheda.

Oggi, dopo anni di miglioramenti e diverse versioni di hardware e software, Arduino è senza dubbio il punto di riferimento in tutto il mondo per appassionati, hobbisti, studenti e non solo: un esempio di hardware open source di successo.

Cos’è Arduino

Ma cos’è Arduino? E a cosa serve? Una scheda Arduino non è altro che un piccolo, semplicissimo computer, neppure lontanamente paragonabile per prestazioni e potenza di calcolo a quelli che siamo abituati a tenere sulla scrivania e a utilizzare ogni giorno. La caratteristica fondamentale della scheda Arduino è la sua capacità di interagire con l’ambiente circostante: mentre i classici computer sono in grado solamente di restare in attesa di comandi impartiti dall’utente, una scheda Arduino collegata ai giusti componenti e opportunamente programmata può interagire con l’esterno e le persone con modalità anche piuttosto lontane dai tradizionali mouse e tastiera.

La scheda dispone di vari pin a cui puoi collegare *sensori* e *attuatori*. I sensori sono componenti elettronici in grado di percepire qualcosa dall’ambiente che li circonda e trasformarlo in un segnale elettrico interpretabile dal processore della scheda Arduino. Esistono sensori di ogni tipo, dai più complessi ai più elementari: immagina un sensore di luce, che registra il livello di luminosità di una stanza, o anche un semplice pulsante, che comunica al processore quando viene premuto dall’utente.

Ci sono sensori che imitano i sensi umani (sono per esempio in grado di reagire a cambiamenti di luminosità o di percepire suoni), ma non solo: in alcuni casi i nostri sensi possono essere superati dall’elettronica sia per precisione (un sensore di temperatura può restituire un dato oggettivo esatto, non una semplice sensazione di “caldo” o “tiepido”), sia per capacità (il nostro organismo non può percepire la presenza di un gas come il monossido di carbonio nell’aria, ma un sensore adeguato non ha problemi a dare l’allarme se lo rileva).

Gli attuatori sono l’esatto opposto: trasformano i segnali elettrici che ricevono in fenomeni percepibili nell’ambiente dagli utenti. Immagina un LED (che emette luce), un altoparlante (che emette suoni) o un motore (che genera movimento).

Sensori e attuatori comunicano con il *processore*, che interpreta gli input dei sensori e determina il comportamento degli attuatori in base alle istruzioni con cui è stato programmato.

Appositi componenti permettono inoltre ad Arduino di comunicare con altri apparati elettronici localmente (Bluetooth, NFC…) o in Rete (Ethernet, Wi-Fi…) per dotare la scheda di possibilità ancora più interessanti.

Le schede Arduino, oltre al processore, incorporano i componenti necessari per semplificare al massimo l’alimentazione dei circuiti e la comunicazione via USB con

il computer per la programmazione e il trasferimento di dati.

> **NOTA** *Per darti un'idea delle prestazioni di Arduino Uno, il punto di riferimento della famiglia di schede, considera che il processore ATmega328 ha 32 KB di memoria flash (utilizzabile per memorizzare lo sketch), 2 KB di RAM e 1 KB di memoria EEPROM (accessibile direttamente dagli sketch per salvare informazioni). Caratteristiche analoghe, ma decisamente inferiori a quelle di un computer da ufficio, non credi?*

Dal lancio dei primi prototipi a oggi il progetto Arduino si è evoluto, diventando un ecosistema sempre più completo che ora può offrire diverse schede, ognuna con le proprie peculiarità e caratteristiche specifiche. L'attenta progettazione ha anche consentito lo sviluppo, da parte del team di Arduino e non solo, di schede aggiuntive, gli *shield*, da innestare sulla scheda principale per dotarla di funzionalità particolari, come per esempio la connessione Ethernet, Wi-Fi o GSM (**Figura I.2**).

La licenza open source

Uno dei fattori che ha contribuito al successo della piattaforma Arduino è stato il rilascio di tutto il materiale relativo alla realizzazione della scheda sotto licenza open source. Il sito ufficiale del progetto (`http://arduino.cc`) mette a disposizione gli schemi elettrici della scheda, che chiunque può scaricare e modificare secondo le proprie necessità. Come per altri progetti open, l'unico vincolo è il rilascio di qualsiasi prodotto derivato sotto una licenza analoga.

Figura I.2 La scheda Arduino Uno con lo shield Ethernet.

Questo approccio ha permesso la nascita di alcuni progetti derivati da Arduino e la commercializzazione di schede basate sull'originale, ma con caratteristiche specifiche per alcune applicazioni. È la community stessa che individua le lacune, propone miglioramenti e suggerisce la strada da seguire per i futuri sviluppi del progetto.

Anche l'IDE Arduino, il software di sviluppo utilizzato per creare gli sketch e

programmare la scheda, è distribuito con licenza open: chiunque, con le giuste competenze, può modificarlo per adattarlo a scopi specifici o contribuire attivamente allo sviluppo delle versioni future mettendosi in contatto con gli altri sviluppatori della community tramite la mailing list dedicata su Google Gruppi.

Queste premesse permettono ad Arduino di essere un progetto sempre alla portata di tutti gli appassionati, in continua evoluzione e ricco di contributi da parte dell’attiva comunità degli utenti.

A chi si rivolge questo libro

Questo piccolo manuale si rivolge a chi è incuriosito dall’elettronica e vuole sperimentare in sicurezza e semplicità muovendo i primi passi nel mondo di Arduino. Partendo dalla spiegazione dei principi della programmazione e della prototipazione con Arduino, verranno illustrati i concetti base e descritte alcune applicazioni della scheda seguendo una serie di esempi pratici.

Di che cosa hai bisogno

Per mettere in pratica gli esempi proposti nel libro avrai bisogno di alcuni componenti esterni, da connettere alla scheda e da far interagire con gli sketch (i programmi) caricati sul processore. Di seguito trovi l'elenco del materiale che utilizzeremo.

> **NOTA** *La maggior parte dei componenti qui elencati è inclusa nei vari kit base disponibili in molti store online. Se ti stai avvicinando ora alla prototipazione elettronica e non hai a disposizione nulla, prendine in considerazione uno come punto di partenza per la tua "cassetta degli attrezzi".*

- Scheda Arduino Uno.
- Cavo USB A-B.
- Alimentatore da 9-12V a 1A per collegare la scheda alla rete domestica a 220V.
- Breadboard.
- Cavetti (*jumper*) di varia lunghezza da utilizzare sulla breadboard.
- LED da 5 mm.
- Resistori (diversi valori).
- Pulsanti adatti a essere posizionati sulla breadboard.
- Potenziometro da 10k Ohm.
- Fotoresistenza VT90N2.
- Sensore piezoelettrico.
- Transistor BC547.
- Relè con bobina da 5V.
- Diodo 1N4007.
- Shield Arduino GSM con SIM attiva.
- Sensore di umidità e temperatura DHT11 (o DHT22).
- Sensore di pressione e temperatura BMP180 (o BMP085).

Senza dubbio la tua dotazione di attrezzature e componenti crescerà assieme alla tua passione, ma per muovere i primi passi e fare tuoi i concetti base il materiale elencato sarà un buon punto di partenza.

SCHEDE ARDUINO ORIGINALI E CONTRAFFATTE

In seguito all'espansione dell'ecosistema Arduino, oggi sul mercato si trovano riproduzioni di schede realizzate senza autorizzazione. Oltre alla qualità inferiore dei prodotti, con tutti i rischi che ne derivano, l'acquisto di queste schede danneggia l'intero progetto Arduino, poiché toglie risorse a chi investe per portarne avanti lo sviluppo.

Un utente poco esperto può trovarsi in difficoltà nel distinguere questo tipo di prodotti, ma oltre alla regola generale del prezzo (se non è allineato con l'offerta ufficiale spesso si tratta di una contraffazione), trovi tutte le indicazioni per evitare sorprese – come per esempio il colore o i dettagli del logo – all'indirizzo `http://arduino.cc/en/Products/Counterfeit`.

Ricorda che la natura open source del progetto non limita lo sviluppo e la vendita di progetti esterni basati sulla piattaforma, del tutto legittimi. Quello a cui qui si fa riferimento è la commercializzazione non autorizzata di cloni.

Precauzioni

Utilizzando questo tipo di apparecchiature elettroniche è importante mantenere sempre alto il livello di sicurezza, senza mai tralasciare le adeguate precauzioni. Non sottovalutare mai i rischi connessi all’utilizzo di circuiti e componenti elettrici, e tieni sempre ben presente che stai lavorando a tuo rischio e pericolo.

In questo libro

Nel **Capitolo 1** prepariamo l'ambiente di lavoro: vengono illustrati i passaggi necessari per installare l'IDE Arduino e ne viene descritta l'interfaccia, per poi passare al collegamento della scheda e all'upload del primo sketch per acquisire familiarità con le procedure da seguire.

Il **Capitolo 2** è dedicato alla lettura degli input digitali: vengono descritti il collegamento di un pulsante a uno dei pin della scheda e lo sketch per interpretare i valori letti. Viene anche introdotto l'utilizzo del monitor seriale, strumento insostituibile per comunicare con la scheda soprattutto in fase di debug degli sketch.

Il **Capitolo 3** tratta input e output più complessi: non si parla più solo di acceso e spento, come per quelli digitali. Gli output PWM permettono di ottenere livelli di output graduali, compresi tra `0` (completamente spento) e `255` (completamente acceso). Gli input analogici, grazie al convertitore A/D (da analogico a digitale) del microcontrollore, trasformano la lettura del voltaggio sui pin da `A0` ad `A5` in un valore compreso tra `0` (0V) e `1023` (5V).

Gli ultimi due capitoli prendono come pretesto progetti più complessi per analizzare alcuni aspetti avanzati della progettazione con Arduino: una lampada in grado di accendersi e spegnersi con due leggeri tocchi sulla superficie su cui è appoggiata e una semplice stazione meteorologica *stand-alone*.

Fritzing: rappresenta i tuoi circuiti

Le illustrazioni dei circuiti realizzati sulle breadboard che troverai in questo libro sono state create utilizzando Fritzing (http://fritzing.org), un software open source che permette di rendere visivamente, in modo ordinato e intuitivo, i circuiti delle breadboard durante i tuoi esperimenti: un sistema comodo per non andare in confusione e molto utile per condividere con altri i tuoi progetti.

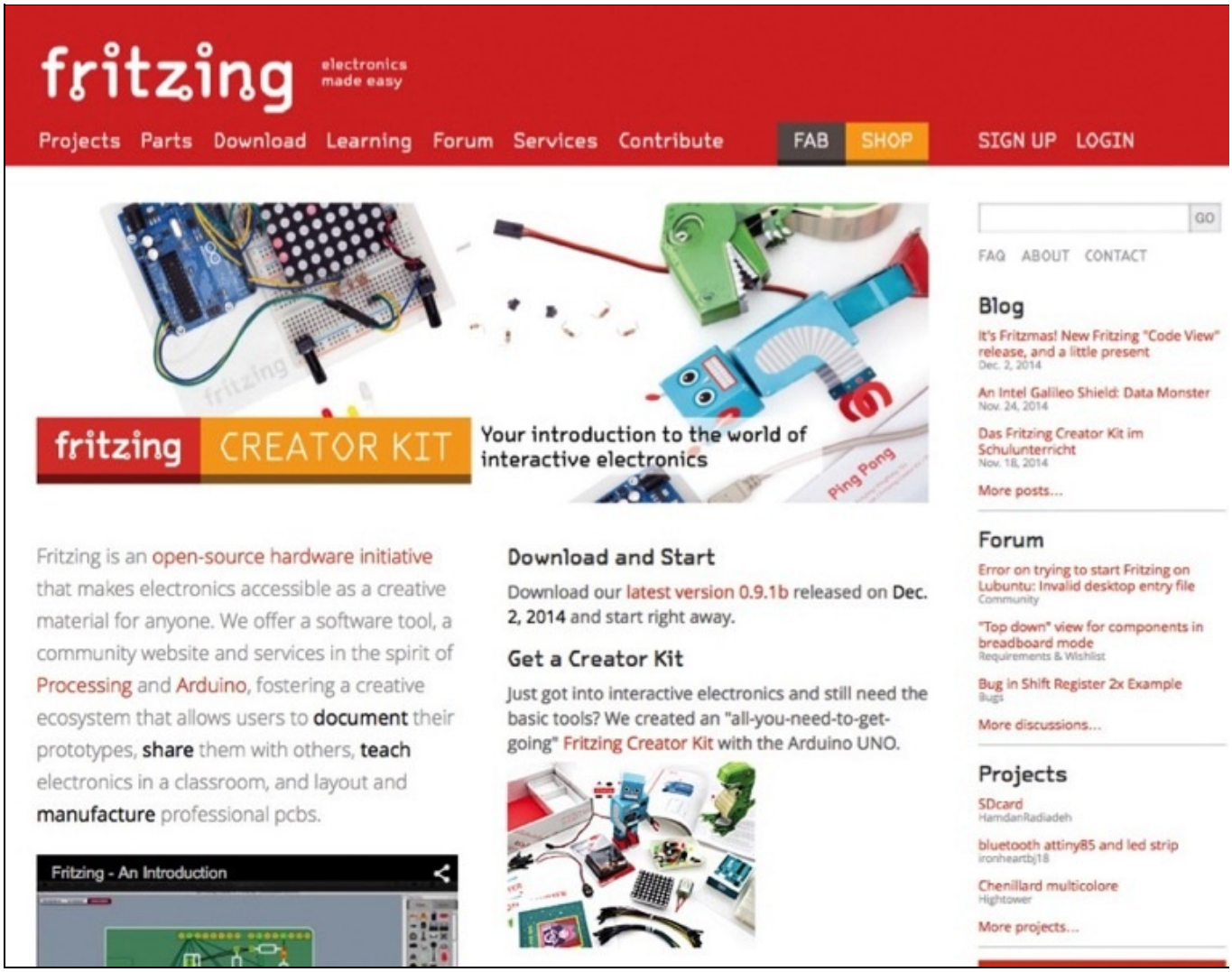


Figura I.3 Con Fritzing visualizzi efficacemente i tuoi progetti Arduino.

La finestra dell'applicazione si divide in due parti principali. Sulla sinistra trovi l'area di lavoro, dove puoi trascinare gli elementi e collegarli tra loro in modo rapido grazie a un'interfaccia *drag & drop*. Sulla destra hai a disposizione la barra degli strumenti con i **Componenti** e l'**Inspector**. Vengono fornite alcune librerie di componenti già pronti per replicare la struttura dei tuoi circuiti e tenere traccia dei progressi mentre lavori.

> **NOTA** *Questo software non sostituisce in alcun modo la realizzazione pratica dei circuiti, di cui non puoi testare il comportamento né simulare l'utilizzo. È esclusivamente uno strumento per rappresentarli graficamente.*

Terminata la fase di prototipazione, quando hai reso definitiva la struttura del circuito che vuoi realizzare, puoi utilizzare le linguette (*tab*) che trovi in alto per muoverti tra le differenti modalità di visualizzazione (**Breadboard**, **Schema** e **PCB**), per progettare persino il circuito stampato e farlo produrre materialmente tramite l'apposito servizio.

NOTA *Tra le tab in alto è stata introdotta anche la voce* Code*, che permette di lavorare sullo sketch direttamente attraverso Fritzing. In ogni caso, per compilare il programma e trasferirlo sulla scheda è indispensabile installare sul computer l'IDE Arduino.*

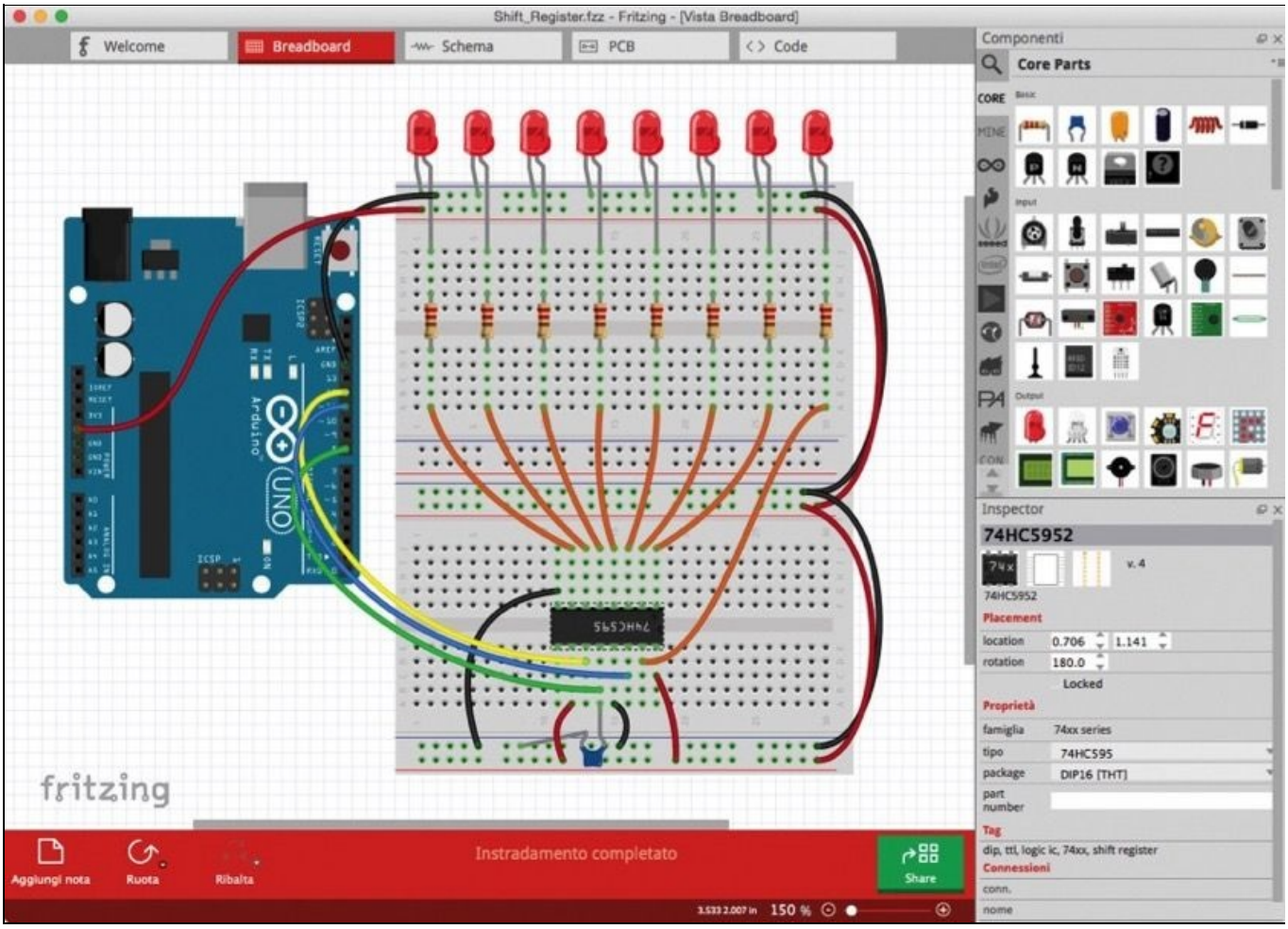


Figura I.4 Uno schema rappresentato nella vista Breadboard.

Tra gli esempi che trovi nel software ci sono molti circuiti che spiegano come collegare i componenti più diffusi alla scheda Arduino in modo corretto. Esplora anche questo materiale per trarre ispirazione ed evitare errori durante i tuoi esperimenti.

Tieni presente che Fritzing, sebbene si adatti in modo perfetto allo scopo, non è pensato solo per lavorare con Arduino. Puoi rappresentare anche circuiti elettrici di altra natura, se necessario, e creare schemi per PCB di qualsiasi tipo.

Il primo approccio ad Arduino

In pochi passaggi potrai configurare l'ambiente di lavoro per iniziare a utilizzare la scheda Arduino Uno: segui la rapida procedura, e in men che non si dica sarai pronto a conoscere la scheda e a mettere in pratica i primi esperimenti.

Per prima cosa è indispensabile scaricare dal sito ufficiale del progetto `http://arduino.cc` il software di sviluppo: seleziona dal menu la voce **Download** o digita direttamente `http://arduino.cc/en/Main/Software` nel browser.

L'IDE Arduino è distribuita per Windows, OS X e Linux. Seleziona il link specifico per avviare il download del pacchetto di installazione del software adatto.

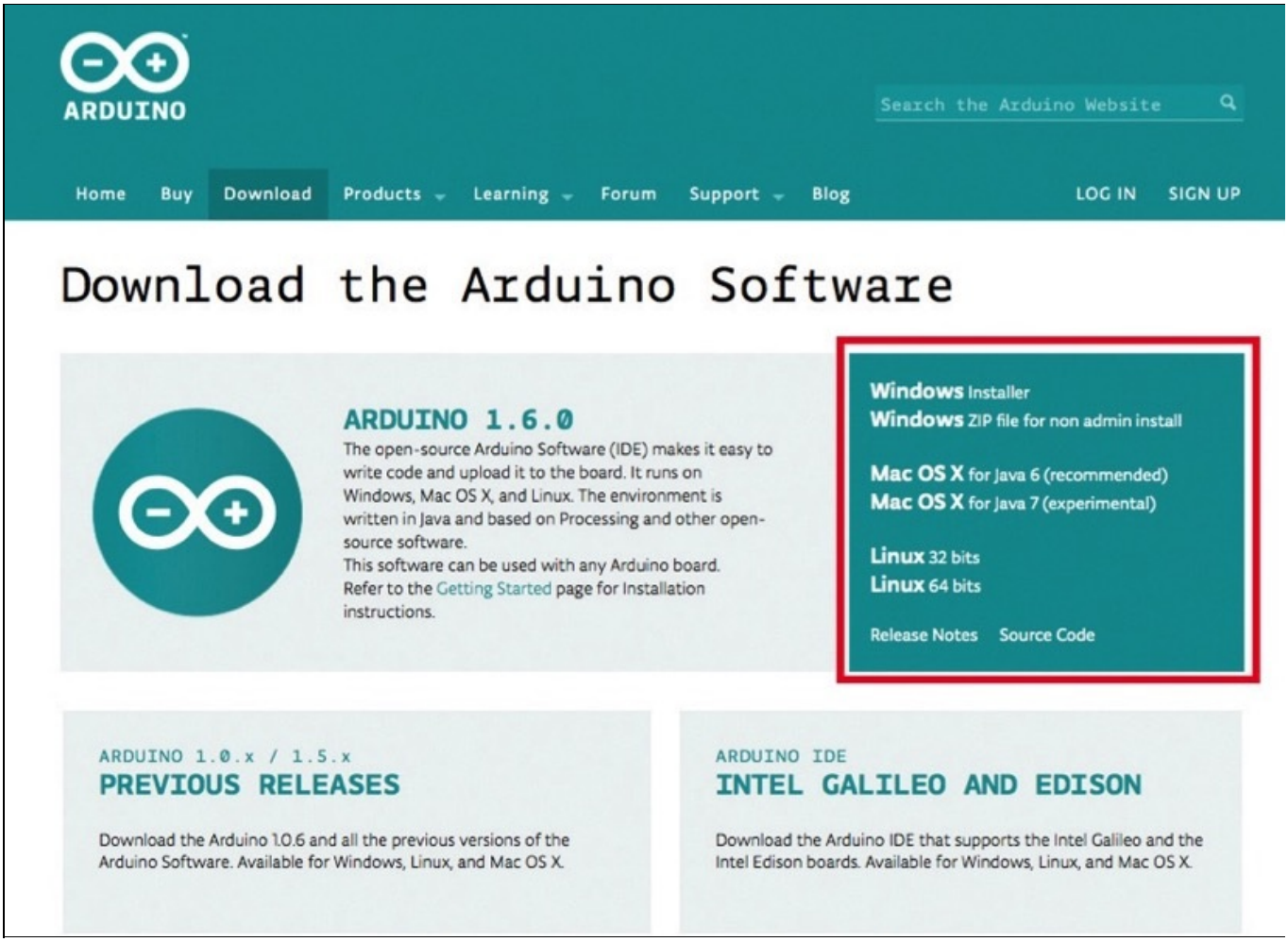


Figura 1.1 La pagina di download del software Arduino: scegli la versione adatta in base al sistema operativo che utilizzi.

NOTA *La release 1.6.0 dell'IDE – a cui si fa riferimento nel manuale salvo dove diversamente specificato – rappresenta un punto di svolta significativo: vengono unificate le versioni 1.0.x e*

1.5.x, prima sviluppate in parallelo. Ora una sola applicazione ti permette di interfacciarti anche con Arduino Yún e Due.

Prepara l’ambiente di lavoro

Per iniziare a sperimentare con Arduino il primo passo consiste nell’installare il software e far sì che il sistema operativo riconosca l’hardware. La procedura è semplice; negli esempi passo passo prenderemo in considerazione la scheda Arduino Uno e le versioni più aggiornate dei diversi sistemi operativi. In caso di configurazioni differenti consulta il sito `http://arduino.cc`, dove puoi trovare molte risorse.

Windows

Terminato il download del pacchetto **Windows Installer**, fai doppio clic sul file `.exe` e segui la procedura di installazione del software. Ti verrà richiesta la password di amministratore e una ulteriore conferma di sicurezza riguardante l’autore dei driver della scheda.

A questo punto collega la scheda Arduino Uno a una porta USB del computer: la scheda verrà riconosciuta in modo automatico e sarà pronta per essere utilizzata.

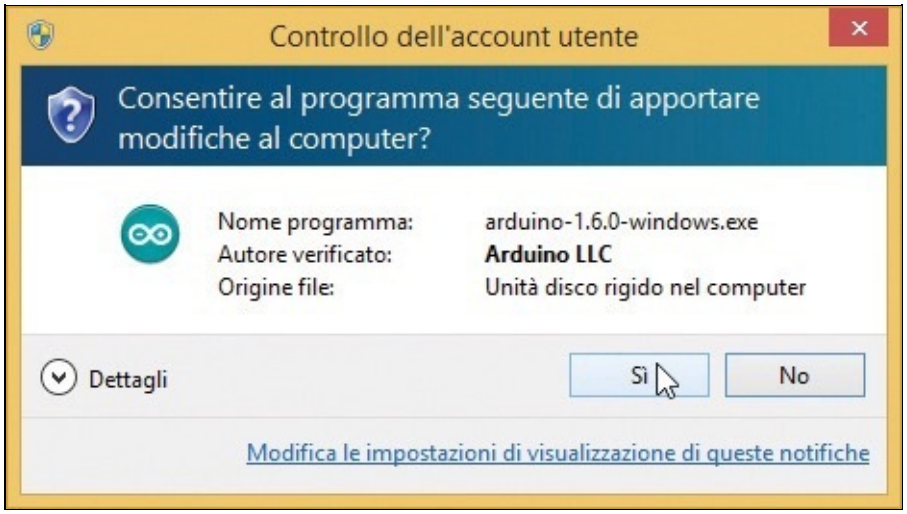
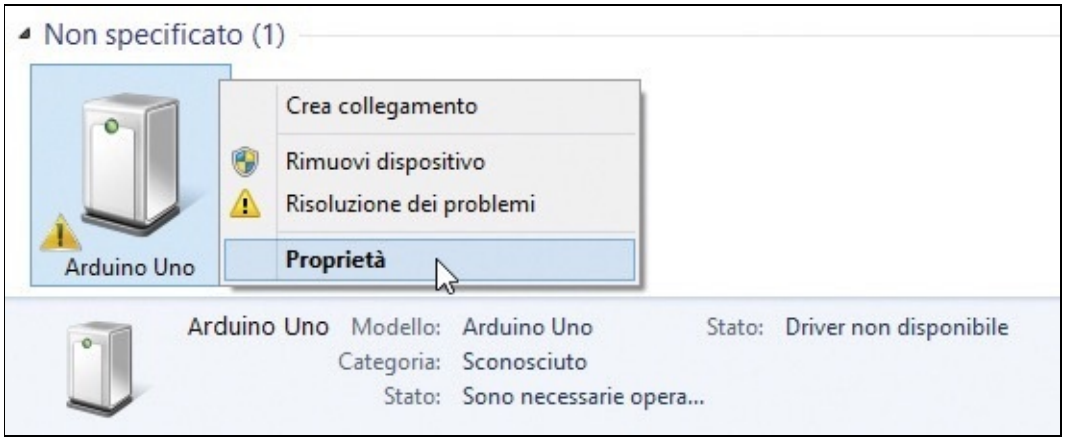


Figura 1.2 È necessario autenticarsi come amministratore durante la procedura di installazione del software Arduino.

INSTALLAZIONE MANUALE DEI DRIVER

In caso di problemi, o se vuoi utilizzare una scheda meno recente, dal **Pannello di controllo** di Windows puoi visualizzare la periferica, con l’icona gialla di allerta che ne indica il mancato riconoscimento, e installare manualmente i driver inclusi nel software Arduino.



Fai doppio clic sulla periferica per visualizzarne le proprietà e all'inizio della procedura guidata di aggiornamento dei driver scegli **Cerca il software del driver nel computer**, sfoglia le cartelle fino a raggiungere **drivers** contenuta nella directory **Arduino** installata in precedenza, di default in **C:/Programmi(x86)**, e conferma con **Avanti**.

OS X

Se utilizzi un Mac la procedura è ancora più semplice. Scarica l'archivio ZIP **Mac OS X** ed espandilo. Al suo interno troverai un solo file, nominato `Arduino`: trascinalo nella cartella **Applicazioni** (**Applications**) e hai terminato l'installazione. OS X riconosce in modo automatico la scheda, perciò non hai bisogno di eseguire altre operazioni (**Figura 1.3**).

NOTA *Una delle migliorie introdotte con Arduino Uno è l'utilizzo di un chip Atmega16U2 o Atmega8U2, più versatile del precedente, per la gestione della connessione USB con il computer. Le schede di vecchia generazione sono dotate di un chip FTDI che assolve a questa funzione. Se stai utilizzando una di queste schede, devi scaricare dal sito del produttore e installare i driver relativi a quel chip (*`http://www.ftdichip.com/Drivers/VCP.htm`*).*

Figura 1.3 Trascina l'icona dell'IDE Arduino nella cartella Applicazioni (Applications) per completare l'installazione.

Linux

Anche con Linux puoi utilizzare l'IDE di sviluppo e programmare le schede Arduino. L'Installazione può variare in base alla configurazione del sistema e alla distribuzione: sul sito ufficiale del progetto trovi alcune guide specifiche in lingua inglese, partendo dall'indirizzo `http://www.arduino.cc/playground/Learning/Linux`.

> **NOTA** *La procedura generica prevede in ogni caso l'installazione dell'ambiente Java openjdk-7-jre e dell'IDE scaricata dal sito di Arduino: a partire dalla versione 1.0.1 del software, i tool necessari per la compilazione degli sketch e il loro trasferimento sulla scheda sono inclusi e non è necessario installarli a parte.*

In Ubuntu puoi affidarti all'**Ubuntu Software Center**, che semplifica al massimo l'operazione: cerca **Arduino** e fai clic su **Installa**: inserisci la password di amministrazione e attendi la conclusione del processo automatizzato.

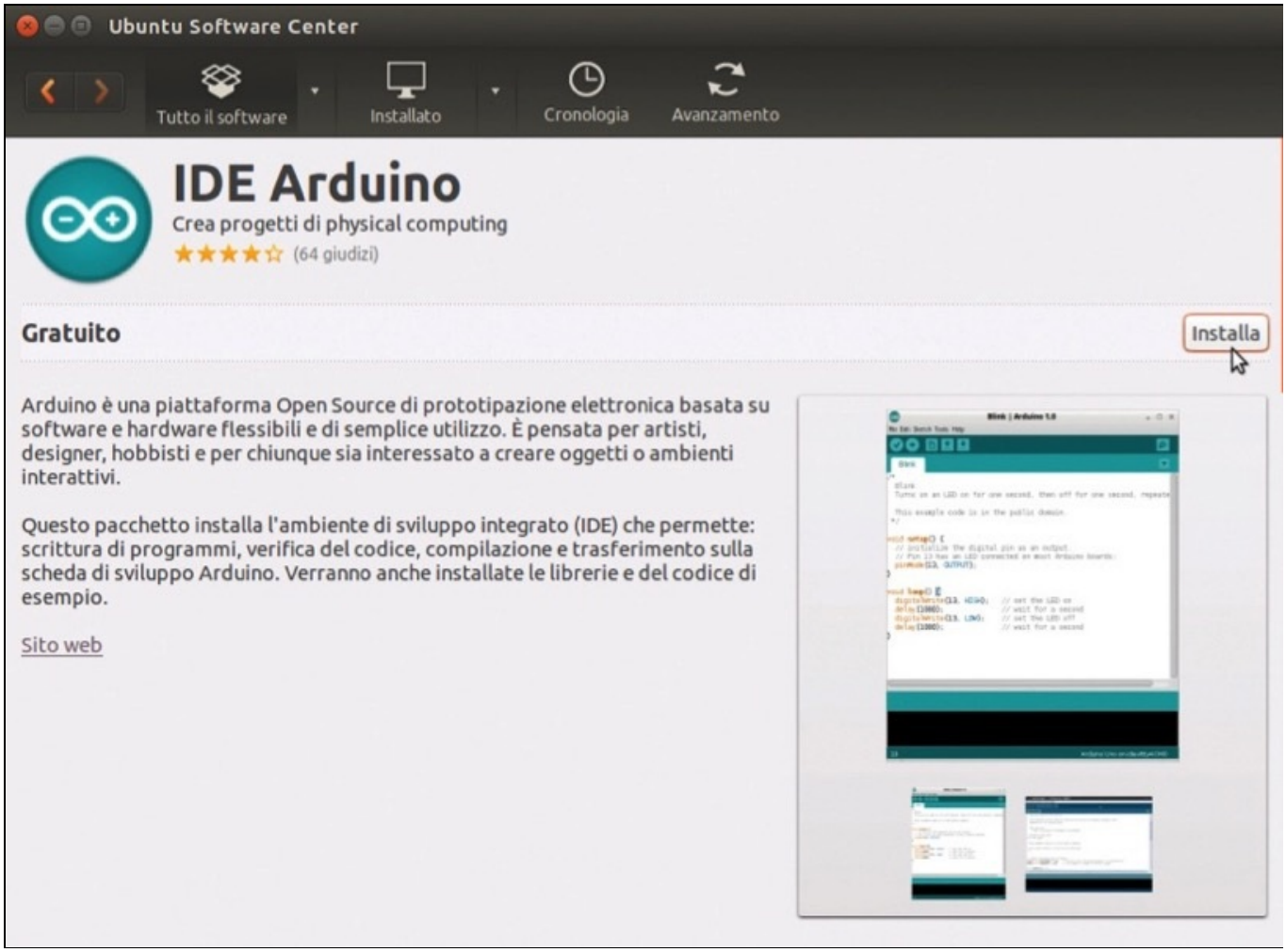


Figura 1.4 Fai clic su Installa per avviare il download del software.

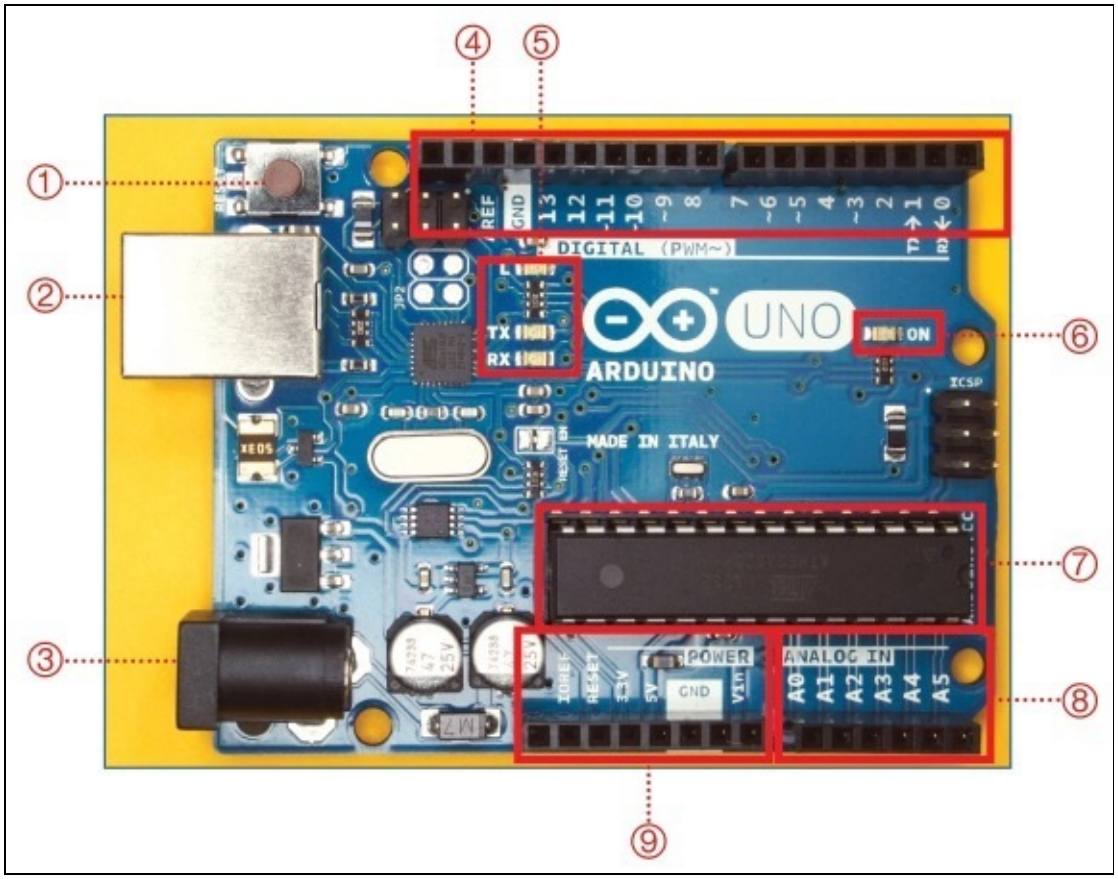
NOTA *La versione del software disponibile tramite l'*Ubuntu Software Center *potrebbe non essere la più aggiornata; verifica i dettagli.*

La procedura guidata si occuperà anche di aggiungere il tuo utente al gruppo **dialout**, operazione necessaria per consentire la comunicazione USB tra il computer e la scheda.

Connetti la scheda

Ora che il software è installato nel sistema, per caricare uno sketch è sufficiente utilizzare la connessione USB: il software Arduino riconoscerà l'hardware e ne permetterà la programmazione.

Prima di esplorare l'IDE e iniziare con i primi esperimenti, facciamo un passo indietro e identifichiamo alcuni dei componenti saldati sulla scheda.

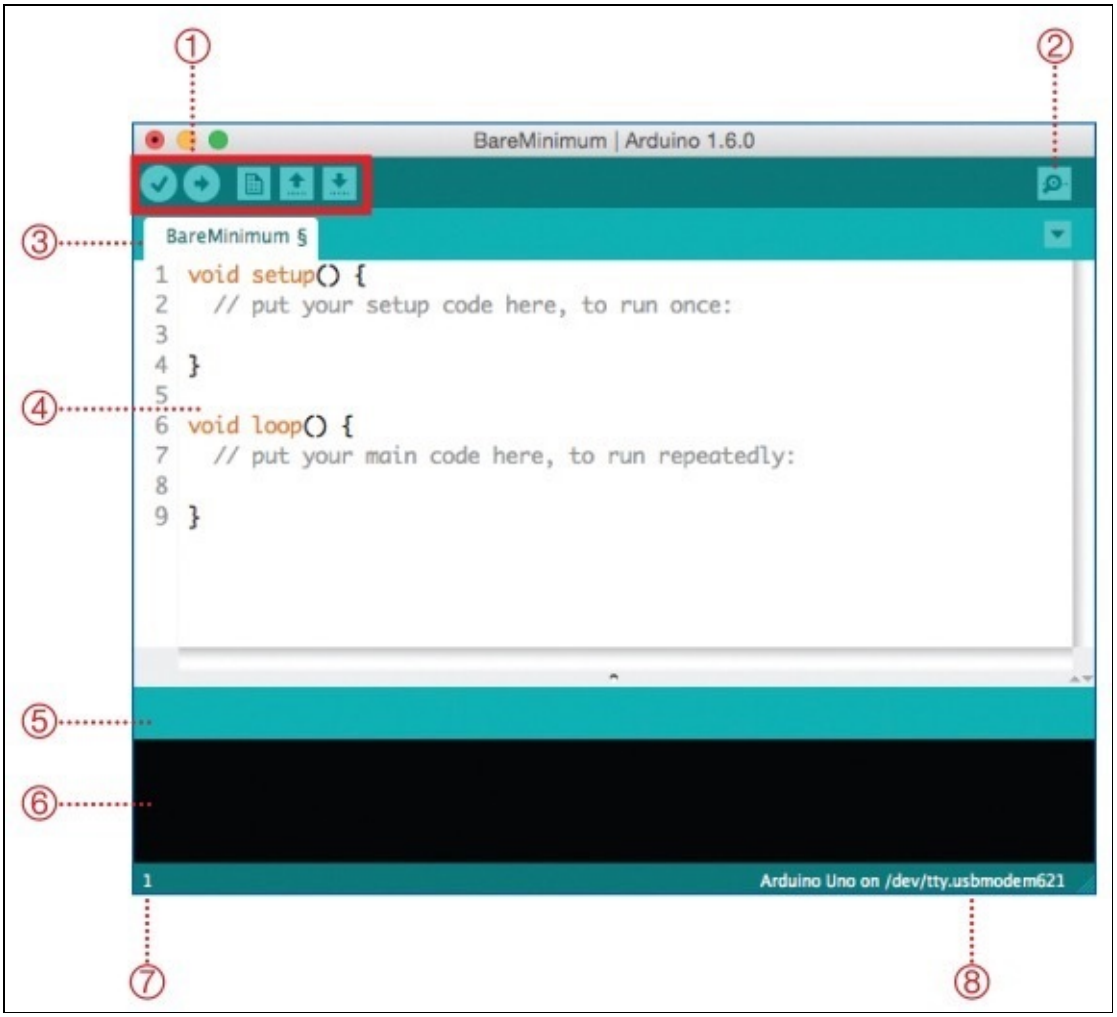


1. Il tasto di reset.
2. Il connettore USB.
3. Il jack di alimentazione alternativa.
4. I pin di input/output digitale (quelli identificati dal simbolo ~ sono utilizzabili in modalità PWM).
5. I LED integrati sulla scheda: quello contrassegnato da L è collegato al pin 13, gli altri due segnalano le comunicazioni seriali in ingresso (RX) e in uscita (TX).
6. Il led ON si illumina quando la scheda riceve alimentazione.
7. Il "cuore" della scheda: il processore centrale.
8. I pin di input analogico.
9. I pin dedicati all'alimentazione: voltaggi predefiniti e pin GND.

NOTA *L’alimentazione per il funzionamento della scheda è fornita direttamente tramite il cavo USB. In alternativa, per esempio per progetti in cui la scheda è indipendente dal computer, puoi connettere un alimentatore o un pacco batteria al jack indicato al punto 3 nella figura. Arduino Uno sceglierà automaticamente la sorgente disponibile.*

L’IDE Arduino

A questo punto siamo quasi pronti per iniziare a sperimentare: scopriamo gli elementi della schermata principale dell’IDE (acronimo di *Integrated Development Environment*, ambiente di sviluppo integrato).

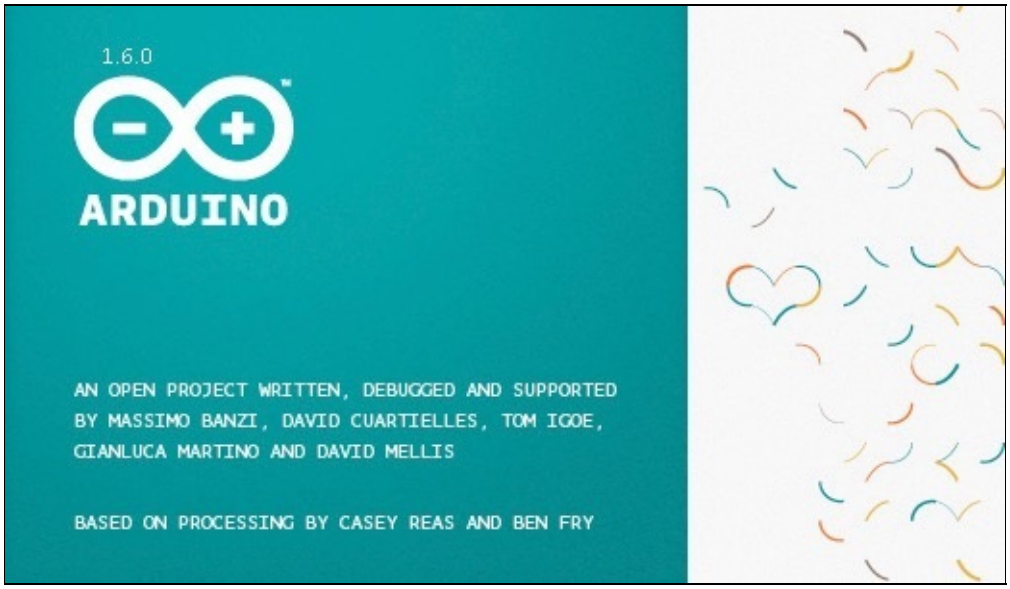


1. I pulsanti per accedere alle funzioni più importanti.
2. Il pulsante per visualizzare il monitor seriale.
3. L’area dedicata alle tab con i file dello sketch aperto: funziona in modo analogo a quella di un browser web.
4. L’area principale dell’editor testuale, dove scrivere il codice che verrà poi compilato per programmare la scheda.
5. La barra di notifica, dove il software ti aggiorna sulle operazioni in corso.
6. La console, dove vengono visualizzati messaggi di errore e altre informazioni.
7. Il numero della riga su cui è posizionato il cursore.
8. Nella barra in basso sono sempre visualizzate la scheda selezionata e la porta di comunicazione utilizzata.

Nonostante a prima vista l’interfaccia sembri piuttosto scarna, ti accorgerai che dispone di tutti gli elementi necessari per i tuoi esperimenti. La semplicità è senza dubbio uno degli elementi chiave che contribuiscono al successo della piattaforma Arduino.

INFORMAZIONI E CREDITI

Nella parte alta della finestra dell’IDE trovi il nome dello sketch e la versione esatta del software Arduino che stai utilizzando. A volte questo dettaglio può esserti utile, per esempio quando vuoi segnalare un bug dell’applicazione al team di sviluppo.



Anche nella schermata di benvenuto del programma trovi questa informazione (sopra al logo) e i crediti.

LE ORIGINI DELL’IDE

L’IDE Arduino prende ispirazione da quella di Processing, un linguaggio di programmazione open source utilizzato soprattutto per lo sviluppo di effetti grafici, computer art o contenuti interattivi. Trovi maggiori informazioni e approfondimenti sul sito ufficiale del progetto all’indirizzo http://processing.org.

La barra degli strumenti

Gli strumenti principali sono sempre a disposizione, nella parte alta dell’interfaccia.

Figura 1.5 La barra degli strumenti.

Vediamo nel dettaglio, da sinistra verso destra, la funzione di ciascun pulsante.

- **Verifica**: esegue un controllo del tuo sketch in cerca di errori. Il risultato viene visualizzato nella barra di notifica in basso, nella quale puoi seguire anche lo stato di avanzamento del processo. Se lo sketch è corretto viene visualizzata la scritta **Compilazione terminata**, altrimenti vengono mostrati dettagli sull'errore riscontrato, e la riga incriminata dello sketch viene evidenziata.
- **Carica**: è il pulsante su cui fare clic per caricare lo sketch compilato sulla scheda. In ogni caso questo comando esegue il controllo preliminare dello sketch: non sarà possibile trasferire sulla scheda uno sketch difettoso o mal funzionante.
- **Nuovo**: apre uno sketch vuoto. È il punto di partenza per un nuovo progetto.
- **Apri**: permette di accedere agli sketch di esempio dello sketchbook e a quelli salvati da te. La prima voce del menu, **Apri**, ti dà accesso alla classica finestra di dialogo del sistema operativo per individuare e aprire un qualsiasi file.
- **Salva**: salva le modifiche allo sketch corrente su disco.
- **Monitor seriale**: apre la finestra del monitor seriale, dove puoi leggere i dati trasmessi dalla scheda Arduino sulla porta seriale e comunicare a tua volta con la scheda. Approfondiremo l'utilizzo di questo strumento più avanti.

Ricorda che hai sempre a disposizione la barra dei menu in alto, che nelle cinque voci **File**, **Modifica**, **Sketch**, **Strumenti** e **Aiuto** raccoglie tutti i comandi e le funzioni del software.

NOTA *Una funzione davvero utile, accessibile dal menu* Strumenti*, è* Formattazione automatica*, che riordina il tuo sketch uniformandone l'indentazione e rendendolo più semplice da "leggere".*

Le preferenze dell'applicazione

Dal menu **File** (per Windows e Linux, dal menu **Arduino** per OS X) puoi accedere a un pannello che raccoglie le principali impostazioni utili per la personalizzazione del software.

Potrebbe servirti, soprattutto quando lavorerai su sketch complessi, abilitare la visualizzazione dei numeri di riga: ti aiuteranno a orientarti meglio tra le righe di codice.

Come indicato anche in fondo al pannello, puoi avere un controllo ancora più completo sulle preferenze dell'ambiente di sviluppo modificando il file `preferences.txt`, del quale viene indicata la posizione nel sistema (**Figura 1.6**).

L'ESTENSIONE .INO

Con il rilascio della versione 1.0 dell'IDE è cambiata anche l'estensione con cui vengono salvati su disco gli sketch: ora il progetto Arduino utilizza l'estensione ufficiale `.ino`. In precedenza gli sketch venivano salvati con estensione `.pde`, presa in prestito da Processing, il linguaggio di programmazione open source da cui ha tratto spunto Arduino.

In ogni caso, se necessario, il software si occuperà in modo automatico della conversione chiedendoti conferma.

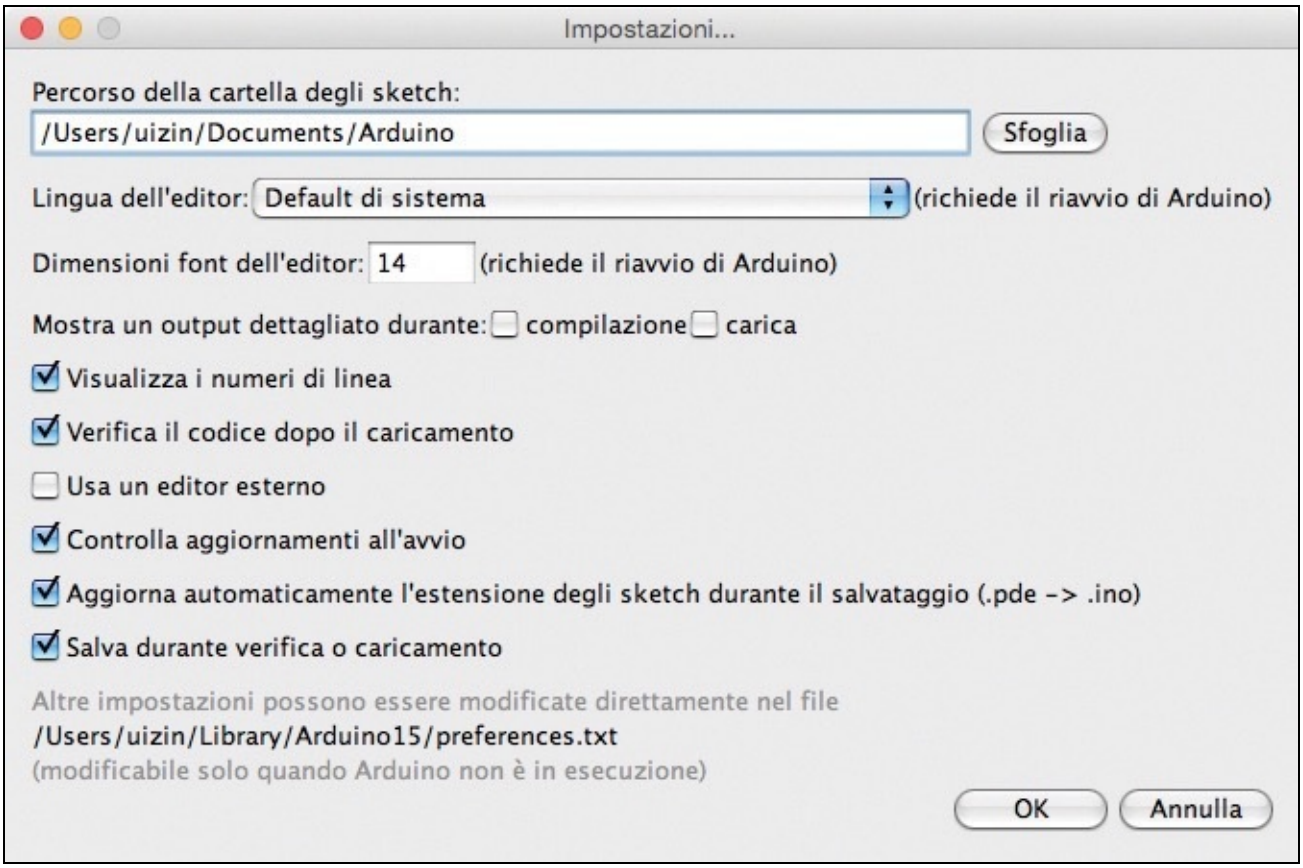


Figura 1.6 Il pannello delle preferenze dell'IDE Arduino.

Il primo sketch: Blink

Ora che abbiamo chiarito quali sono le procedure di configurazione iniziali, è finalmente arrivato il momento di collegare la scheda Arduino alla porta USB del computer e cominciare a sperimentare materialmente.

Il primo sketch che caricheremo sulla scheda farà lampeggiare il LED integrato su di essa, collegato al pin `13`. Questo elementare sketch, detto **Blink**, ci permetterà innanzitutto di assicurarci che la connessione tra scheda e computer sia impostata correttamente.

Avvia l'IDE e collega la scheda Arduino Uno alla porta USB. Per prima cosa dal menu **Strumenti** scegli nell'elenco **Tipo di Arduino** il modello corretto di scheda, nel nostro caso **Arduino Uno**. A questo punto alla voce **Porta seriale** seleziona la connessione seriale corretta.

> **NOTA** *Un metodo piuttosto efficace per capire qual è il nome assegnato alla connessione seriale Arduino dal computer consiste nell'avviare il software Arduino con la scheda non collegata, consultare la lista nel menu* Strumenti > Porta seriale *e in seguito collegare la scheda Arduino. A questo punto consultando nuovamente la lista comparirà una nuova voce, che per esclusione è quella attribuita ad Arduino.*

La sketch **Blink** è disponibile tra quelli di esempio forniti con il software. Fai clic sull'icona **Apri**, nel menu vai su **01.Basics** e selezionalo dall'elenco.

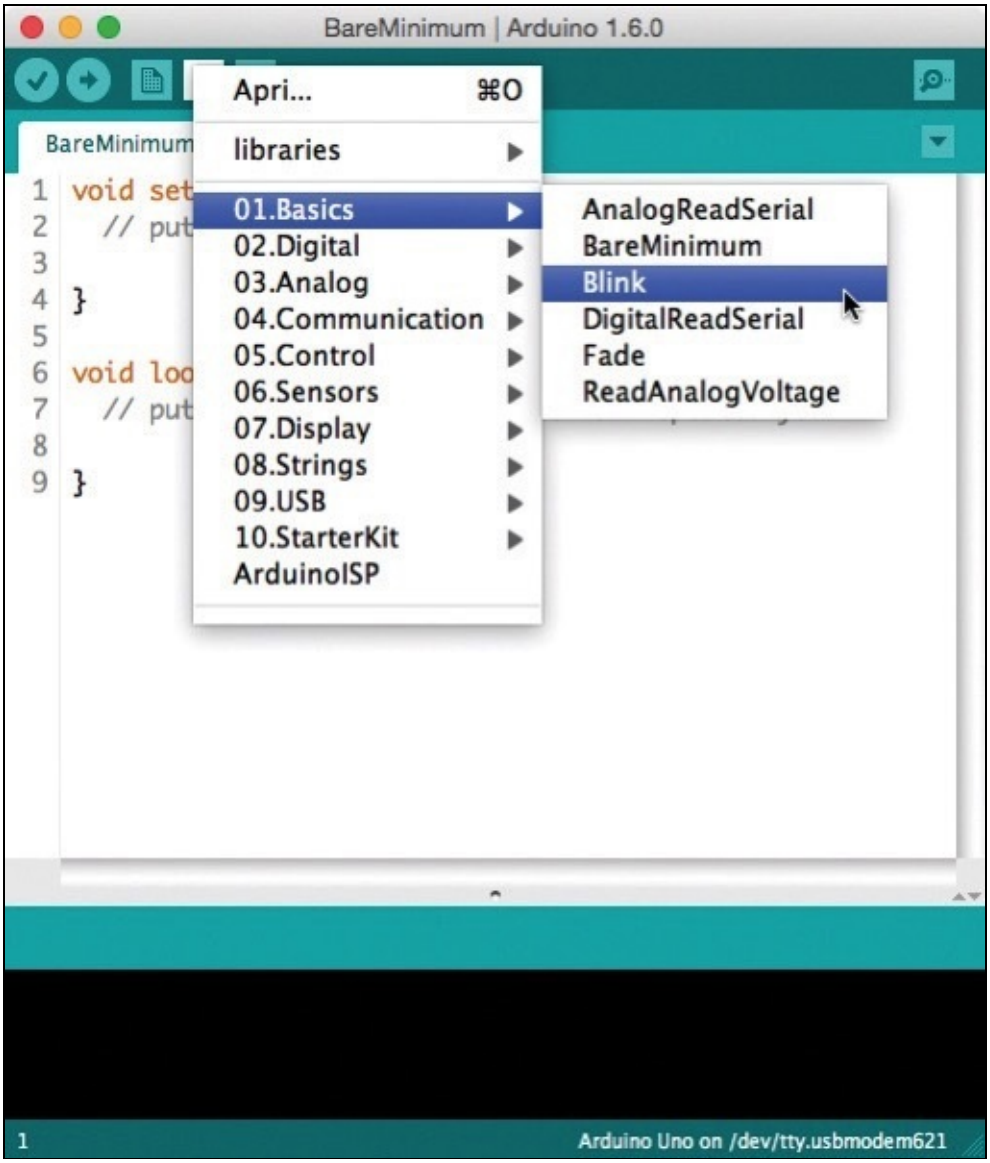


Figura 1.7 Il menu Apri mostra gli sketch di esempio disponibili.

Lo sketch viene caricato nell'editor testuale dell'IDE.

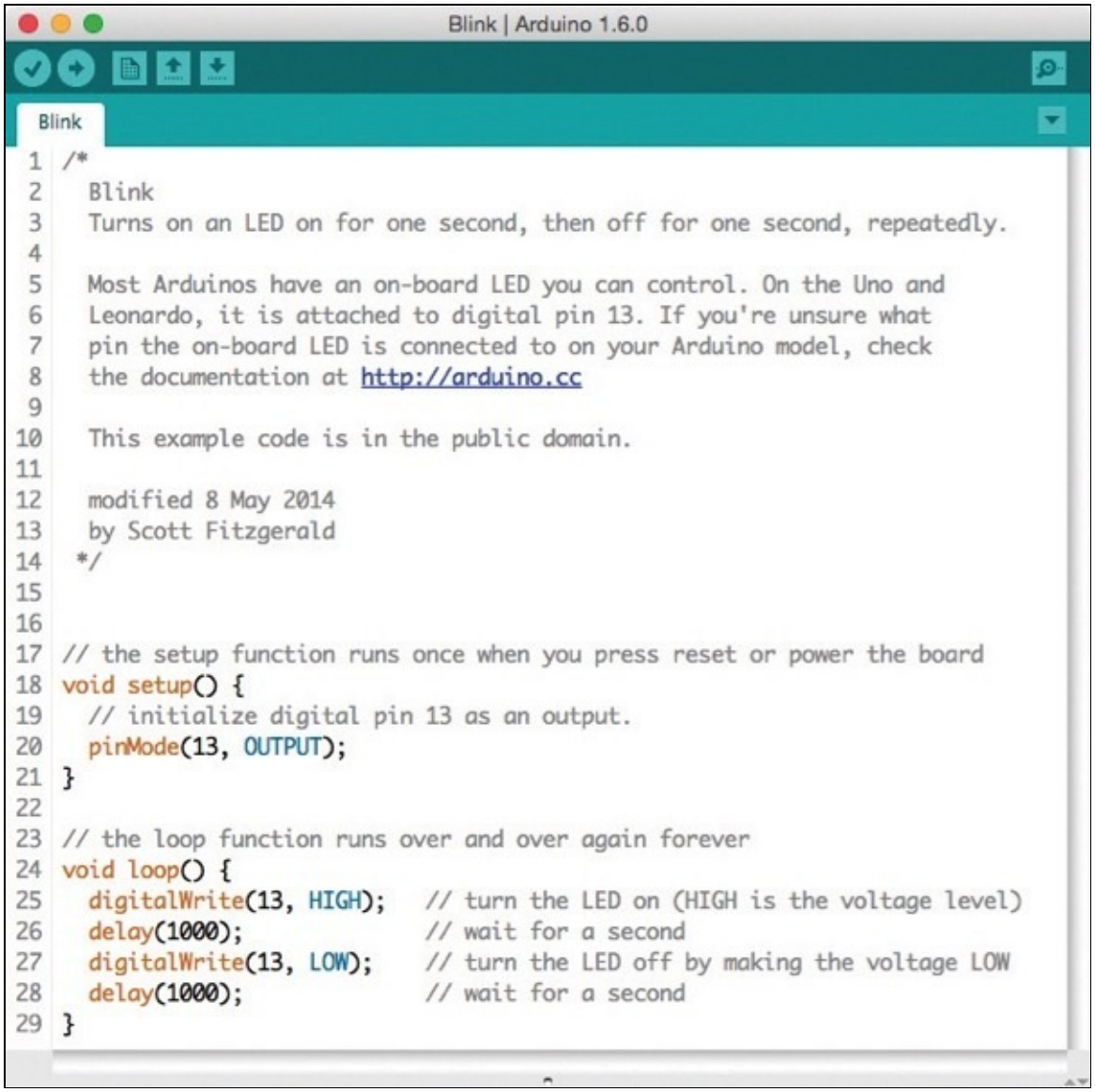


Figura 1.8 Lo sketch Blink.

Senza modificare il codice, fai clic sul pulsante **Carica** in alto. Inizierà il processo di compilazione e trasferimento sulla scheda dello sketch, e nella barra di notifica verrai aggiornato sullo stato di avanzamento.

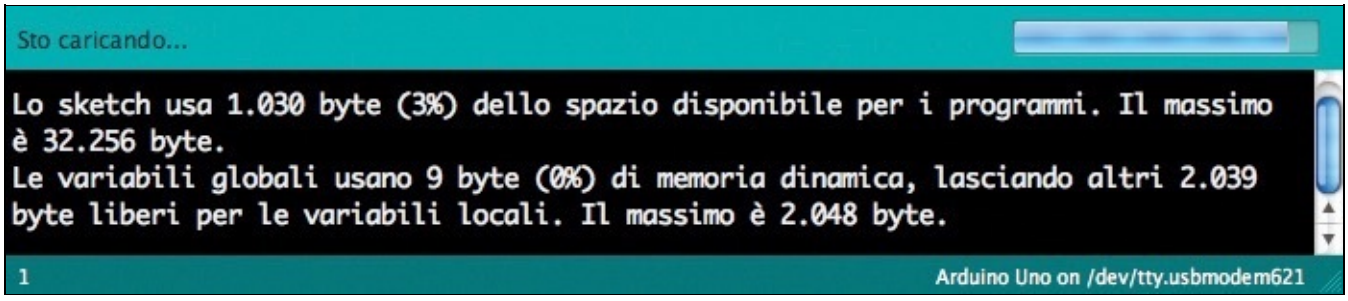


Figura 1.9 La barra di notifica mostra che lo sketch è stato compilato ed è in corso il trasferimento sulla scheda.

NOTA *In caso di problemi durante l'upload, verifica che nel menu* Strumenti *siano selezionati correttamente il modello di scheda Arduino e la porta seriale.*

Durante il trasferimento dello sketch compilato, sulla scheda Arduino vedrai lampeggiare i LED TX e RX, legati all'attività della connessione seriale con il computer. Al termine del processo il LED L inizierà a lampeggiare a intervalli regolari di un secondo.

Passiamo in esame le diverse parti del codice che compone lo sketch, per iniziare a prendere confidenza con la sintassi:

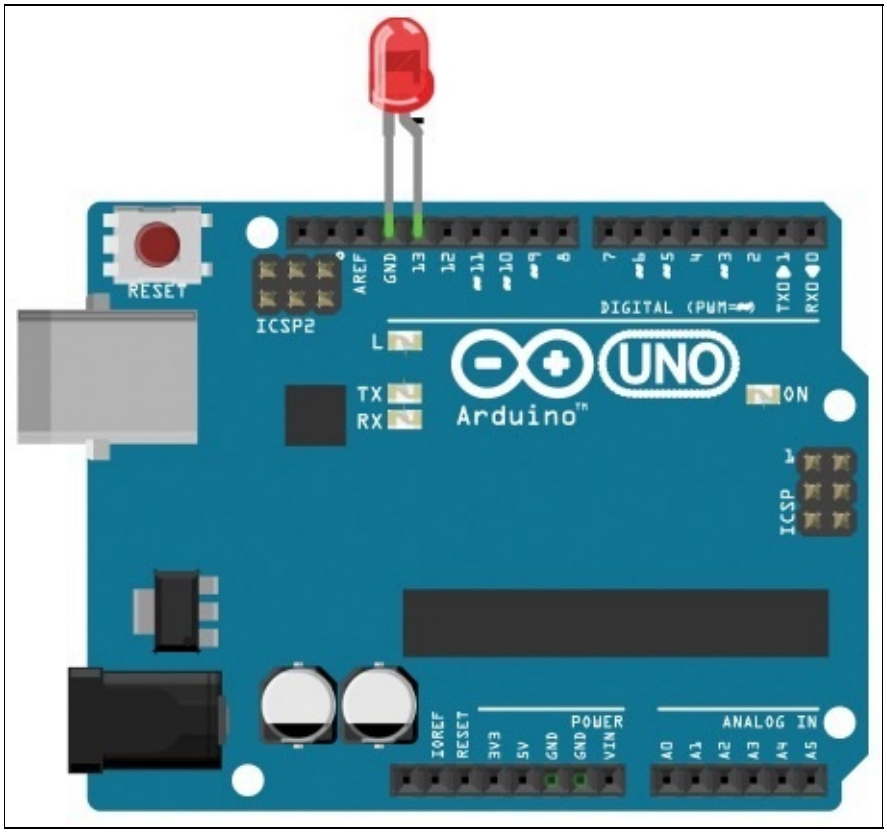
```
/*
  Blink
  Turns on an LED on for one second,
  then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control.
  On the Uno and Leonardo, it is attached to digital
  pin 13. If you're unsure what pin the on-board LED
  is connected to on your Arduino model, check
  the documentation at http://arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
*/
```

Il primo blocco di codice è delimitato da `/*` e `*/`, e di colore grigio nell'editor testuale. Tutto il testo contenuto tra i due marcatori è considerato un commento, ovvero sarà ignorato dal compilatore quando premerai i pulsanti **Verifica** e **Carica**. Puoi utilizzare questa sintassi per aggiungere descrizioni e spiegazioni ai tuoi sketch: in questo modo, quando li condividerai con altri o tornerai a utilizzarli dopo qualche tempo, ti sarà più semplice ricostruirne il funzionamento. Hai a disposizione anche il marcatore `//` per trasformare una porzione di codice in un commento: la differenza rispetto alla sintassi descritta poco fa è che l'effetto resta limitato alla singola riga, senza necessità di un marcatore di chiusura del commento.

> **NOTA** *Per commentare e decommentare velocemente porzioni di codice, seleziona le righe che ti interessano nell'editor e scegli* Commenta > Togli commento *dal menu* Modifica *o dal menu contestuale che appare facendo clic con il tasto destro del mouse.*

IL PIN 13

Puoi effettuare una semplice prova per dimostrare che effettivamente il LED contrassegnato con L sulla scheda è collegato al pin 13: prendi un LED standard, da 5 mm, e infila direttamente i due terminali nel pin 13 e nel pin a fianco, GND. Ricorda che i LED sono diodi, perciò hanno una polarità, e gli impulsi elettrici li possono attraversare solo in una direzione. Riconosci il polo negativo, detto *catodo* – da collegare al pin GND –, dal fatto che è il più corto dei due. Inoltre su molti LED in sua corrispondenza il bordo inferiore, visto dall'alto, è appiattito.



Non appena collegato al pin `13`, il LED esterno si comporterà esattamente come quello integrato sulla scheda.

Continuando a leggere lo sketch, nelle righe successive incontriamo la prima funzione dichiarata, denominata `setup()`:

```
// the setup function runs once
// when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}
```

Tutti gli sketch Arduino devono contenere la definizione di questa funzione. Come indicato anche dal commento in inglese, è la prima che viene eseguita quando la scheda viene alimentata o resettata.

La sintassi per dichiarare una funzione è molto chiara: `void` indica che la funzione non restituisce alcun valore, ma vengono solo eseguite le istruzioni al suo interno (vedi il box più avanti per un approfondimento sui tipi di dato), `setup` è il nome della funzione.

Le parentesi graffe `{` e `}` racchiudono le istruzioni associate alla funzione che stai dichiarando.

NOTA *Dichiarando una funzione hai anche la possibilità di definire dei parametri: devi inserirli tra parentesi tonde dopo il nome della funzione. Nel caso di* `setup` *questo non è previsto.*

Nel caso dello sketch **Blink**, la funzione `setup()` imposta il pin `13` come output. L'istruzione necessaria è `pinMode()`, che accetta due parametri: il primo è il numero del pin digitale al quale vuoi riferirti – nel nostro caso `13` – e il secondo definisce il comportamento: `INPUT` (per collegare sensori) oppure `OUTPUT` (per collegare attuatori).

La funzione `setup()` può diventare più complessa di quella dell'esempio. Sarà utilizzata negli sketch per inizializzare le variabili e definire il comportamento dei pin coinvolti.

I TIPI DI DATO

A ogni variabile è necessario associare il relativo tipo di dato, che ne descrive la tipologia e definisce l'intervallo di valori che tale variabile può assumere. I tipi di dato più comuni sono i seguenti.

- `boolean`: il tipo di dato più semplice, ha valore `true` o `false`.
- `char`: occupa un byte di memoria e contiene un singolo carattere. Puoi anche eseguire operazioni matematiche su questi valori perché ogni carattere equivale a un numero compreso tra –128 e 127, secondo la tabella ASCII.
- `byte` o `unsigned char`: questo tipo di dato ha le stesse caratteristiche di `char`, ma l'intervallo di valori che può assumere va da 0 a 255, senza numeri negativi.
- `int`: lo spazio occupato da questo tipo di dato è di 2 byte, doppio rispetto ai precedenti, e di conseguenza l'intervallo di valori, molto più ampio, equivale ai numeri interi compresi tra –32.768 e 32.767.
- `word` o `unsigned int`: in questo caso le caratteristiche sono analoghe a quelle del tipo `int`, ma i valori che le variabili di questo tipo possono assumere vanno da 0 a 65.535.
- `long`: raddoppiano ancora i byte occupati, diventando 4, e l'intervallo di valori aumenta in modo esponenziale arrivando a comprendere i numeri interi tra –2.147.483.648 e 2.147.483.647.
- `unsigned long`: come per i casi precedenti, questo tipo di dato condivide con `long` tutte le caratteristiche a esclusione dell'intervallo di valori consentiti. Può avere valori compresi tra 0 e 4.294.967.295, senza numeri negativi.
- `float`: anche questo tipo di dato occupa 4 byte, e i valori possibili includono i numeri decimali compresi tra 3,4028235E+38 e –3,4028235E+38. Questo permette, a scapito della velocità di esecuzione, una maggiore accuratezza, per esempio quando viene utilizzato per elaborare informazioni acquisite da sensori esterni.

Il consiglio è di scegliere sempre il tipo di dato adeguato per l'informazione da memorizzare, per migliorare le prestazioni del microprocessore: è inutile per esempio utilizzare un dato `long`, che occupa 4 byte di memoria, per una variabile che sappiamo poter assumere solo valori compresi tra `0` e `10` durante l'esecuzione dello sketch.

NOTA *Dichiarando una funzione, puoi attribuire il tipo di dato* `void` *quando non è previsto che tale funzione restituisca un valore dopo l'esecuzione, come nel caso di* `setup()` *e* `loop()`*. Qualora sia prevista la restituzione di un valore è invece necessario definirne il tipo.*

Osserviamo ora più da vicino la seconda parte dello sketch **Blink**, il contenuto della funzione `loop()`:

```
// the loop function runs over and over again forever
void loop() {
```

```
  digitalWrite(13, HIGH);  // turn the LED on
  delay(1000);             // wait for a second
  digitalWrite(13, LOW);   // turn the LED
  delay(1000);             // wait for a second
}
```

La funzione `loop()` è indispensabile, e le istruzioni in essa contenute vengono ripetute in ciclo continuo quando la scheda è alimentata.

La prima istruzione è `digitalWrite()`, e agisce sul voltaggio del pin `13`, indicato come primo parametro.

> **NOTA** *Come vedi, l'istruzione agisce sullo stesso pin impostato in modalità* `OUTPUT` *con* `pinMode()` *nella funzione* `setup()`.

Il secondo parametro può assumere valore `HIGH`, portando il voltaggio del pin a 5 Volt, oppure `LOW`, portando il voltaggio a 0 Volt. Stiamo attribuendo il valore `HIGH` al pin `13`, perciò il LED si illuminerà.

Nella riga successiva l'istruzione `delay()` interrompe l'esecuzione del codice per un numero di millisecondi pari al valore dell'unico parametro, in questo caso `1000`, ovvero per un secondo.

Dopo un secondo con una nuova istruzione `digtalWrite()` portiamo il voltaggio del pin `13` a 0 Volt con `LOW`, facendo spegnere il LED.

Eseguita l'ultima istruzione, un altro `delay()`, Arduino ripartirà dalla prima, riportando il voltaggio del pin `13` a 5 Volt e riaccendendo il LED in un ciclo senza interruzioni.

Ecco descritto nel dettaglio il codice necessario per far lampeggiare il led `L` sulla scheda Arduino.

LAVORA SCOLLEGATO DAL COMPUTER

Prova a scollegare la scheda dal cavo USB e a innestare un alimentatore con voltaggio compreso tra 7 e 12 Volt nel jack nero (identificato con il numero 3 nella figura che descrive la scheda): Arduino si avvierà e il LED `L` inizierà a lampeggiare come prima. Questo ti dimostra che, quando i tuoi sketch non prevedono interazione con il computer tramite la porta seriale, la scheda è indipendente dalla connessione dati USB e ne sfrutta solo l'alimentazione. In questi casi la trasmissione e la ricezione di dati via USB sono utilizzate per l'upload iniziale dello script e non durante la sua successiva esecuzione.

L’utilizzo delle costanti

Prima di passare a descrivere altre funzionalità dell’ambiente Arduino, soffermiamoci ancora sullo sketch **Blink**, prendendolo come spunto per conoscere una tecnica utile per mantenere leggibili e ordinati i nostri sketch, rendendoli più versatili e semplici da modificare.

Utilizzando le costanti puoi ridurre al minimo i valori *hardcoded*, ovvero inseriti direttamente nel codice. Vediamo come modificare lo sketch di esempio.

Per prima cosa dichiariamo due costanti (anche in questo caso, un commento ci aiuterà ad avere uno sketch più comprensibile):

```
// il pin a cui è collegato il LED
const int ledPin = 13;

// l'intervallo di accensione e spegnimento
const int tempo = 1000;
```

Utilizzeremo la prima costante, `ledPin`, per memorizzare il riferimento del pin a cui stiamo collegando il LED. La seconda, `tempo`, regolerà la pausa tra accensione e spegnimento, espressa in millisecondi.

> **NOTA** *Utilizzando* `const` *per dichiarare una costante stiamo comunicando al compilatore che il valore di tale parametro non verrà mai modificato durante l’esecuzione del programma. Omettendo questa parola chiave si dichiarerà una variabile, il cui valore potrà essere modificato in ogni momento.*

Ora modifichiamo alcuni dettagli della funzione `setup()` per sfruttare le costanti che abbiamo definito:

```
void setup() {
  // inizializza il pin di output
  pinMode(ledPin, OUTPUT);
}

void loop() {
  digitalWrite(ledPin, HIGH);  // LED acceso
  delay(tempo);                // pausa
  digitalWrite(ledPin, LOW);   // LED spento
  delay(tempo);                // pausa
}
```

Nelle istruzioni non sono più inseriti i valori legati al numero di pin e all’intervallo di tempo, ma i riferimenti alle costanti definite in precedenza.

Con i valori attribuiti alle due costanti il comportamento della scheda sarà analogo allo sketch originale, ma ora è possibile cambiare il pin o l’intervallo con molta più comodità.

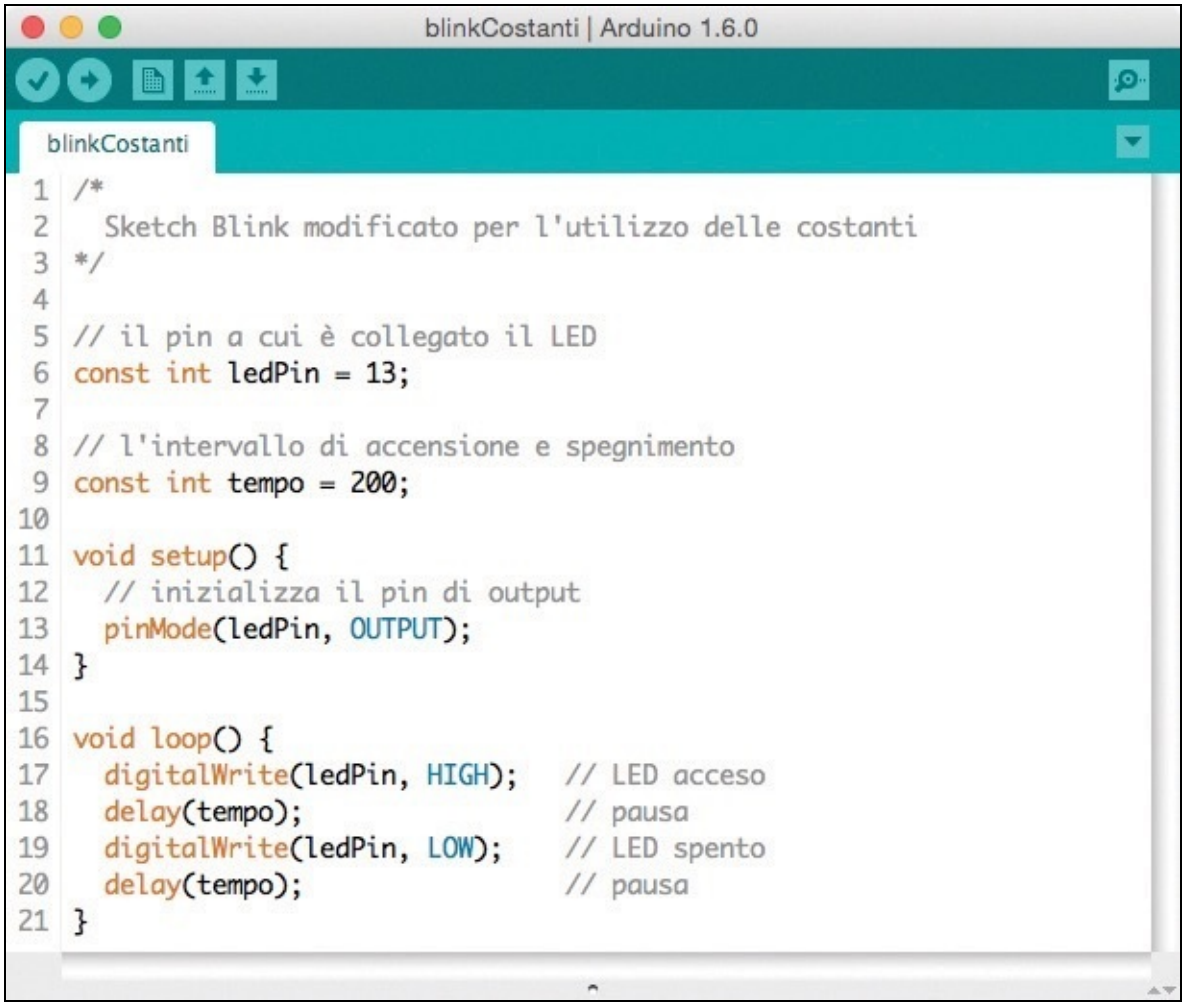


Figura 1.10 Lo sketch nell'IDE Arduino: la colorazione del codice ne aiuta la comprensione.

Prova a modificare il valore della costante `tempo` portandolo a `500`. Fai clic su **Carica**: terminato l'upload sulla scheda, vedrai il LED lampeggiare a un ritmo doppio rispetto a prima. Senza l'utilizzo della costante avresti dovuto modificare il valore nelle singole istruzioni della funzione `loop()`. Immagina di avere uno sketch complesso, composto da centinaia di righe: senza dubbio un approccio del genere permette di intervenire sul comportamento della scheda con molta più comodità evitando errori inutili.

Analogamente puoi comandare un pin diverso dal numero `13` con estrema semplicità, modificando il valore della costante `ledPin`. Questa operazione richiede però la creazione di un circuito per collegare un LED alla scheda, perché come abbiamo detto l'unico pin che ha un LED integrato è il `13`.

Innesta sulla breadboard il LED, facendo attenzione alla sua polarità. Collega un resistore da 220 Ohm (vedi il box alla fine di questo capitolo per capire come individuare il valore dei resistori) alla gamba più lunga del LED (è l'anodo, il polo positivo). Poi con due piccoli cavi, detti *jumper*, collega l'altra estremità del resistore

al pin che vuoi comandare dal software e il polo negativo del LED (il catodo) a `GND`. Nella **Figura 1.11** il LED è collegato al pin `4`.

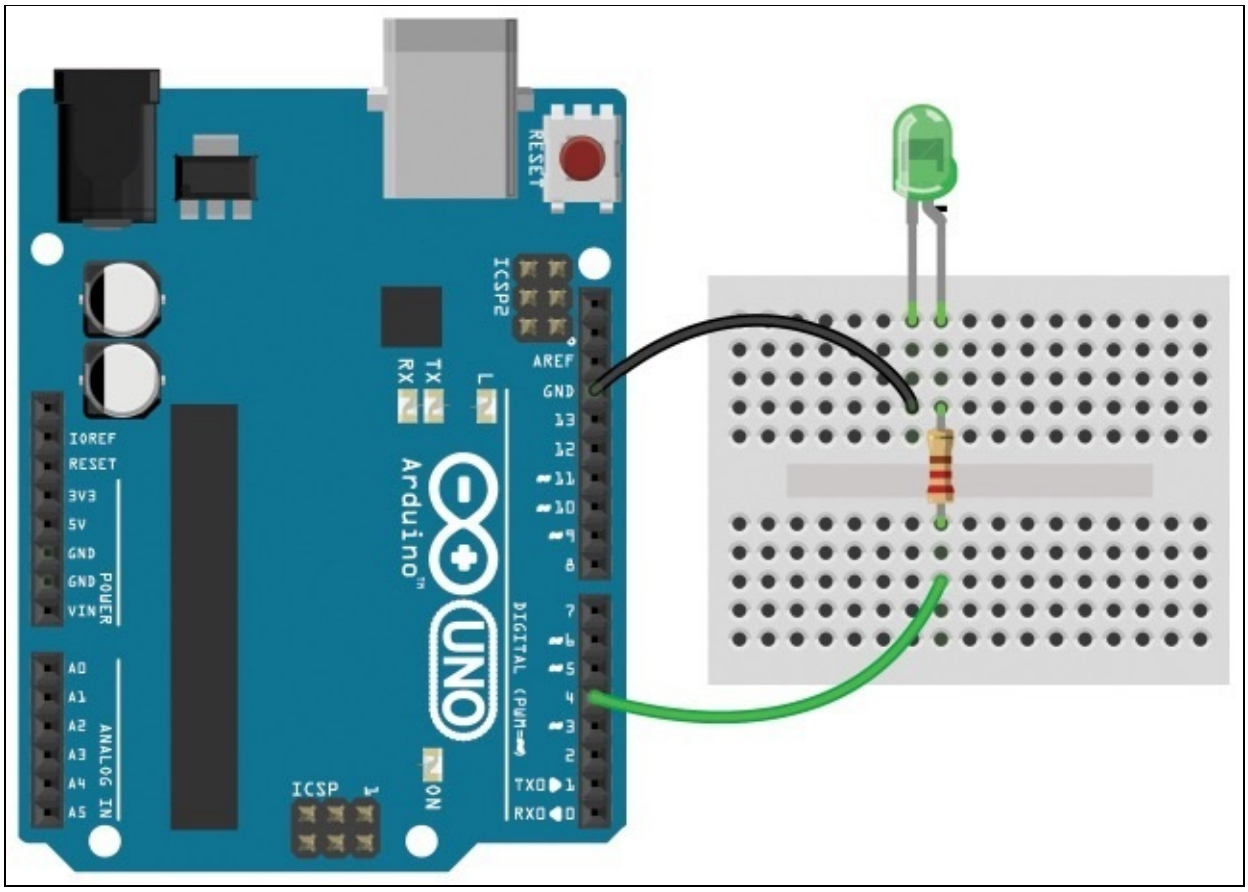


Figura 1.11 Il LED nella breadboard collegato al pin 4 di Arduino.

Nello sketch definisci il valore della costante `ledPin = 4` e fai clic su **Carica**: ammirerai il LED lampeggiare, mentre il led `L` sulla scheda resterà spento.

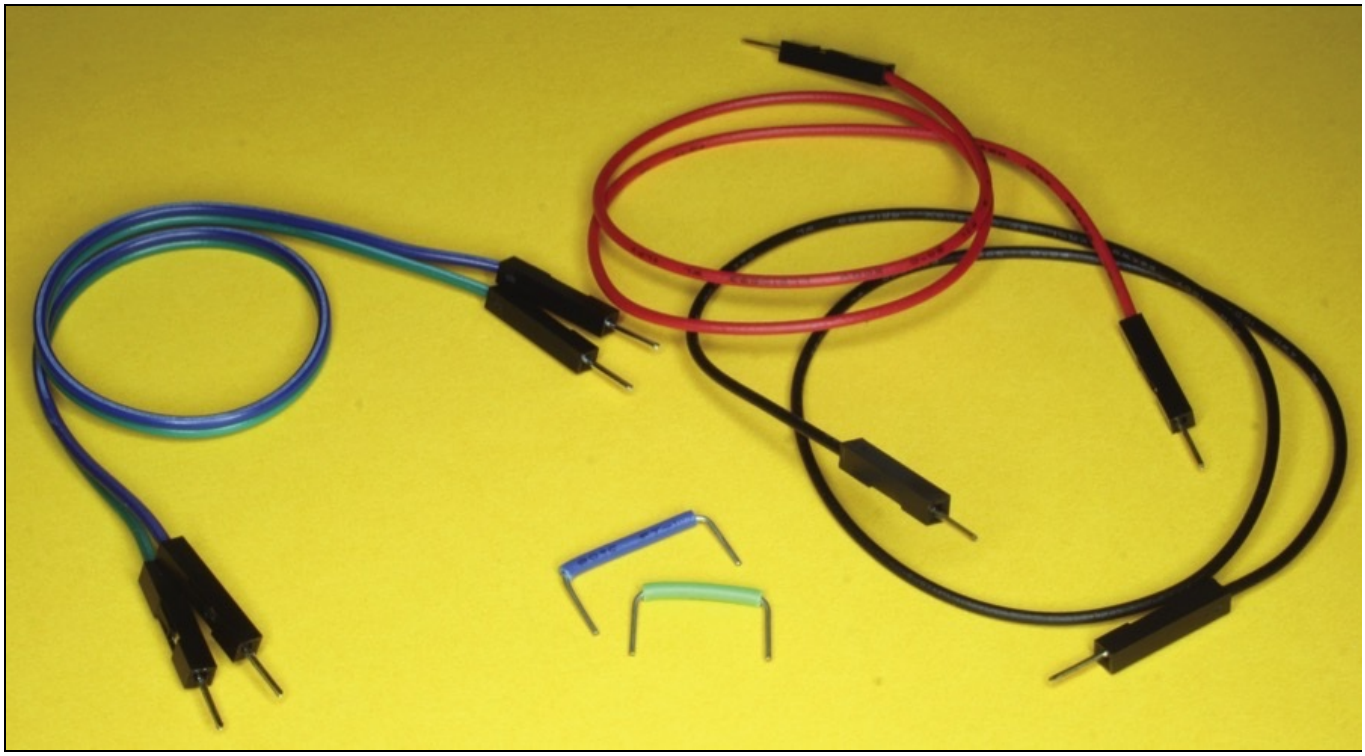
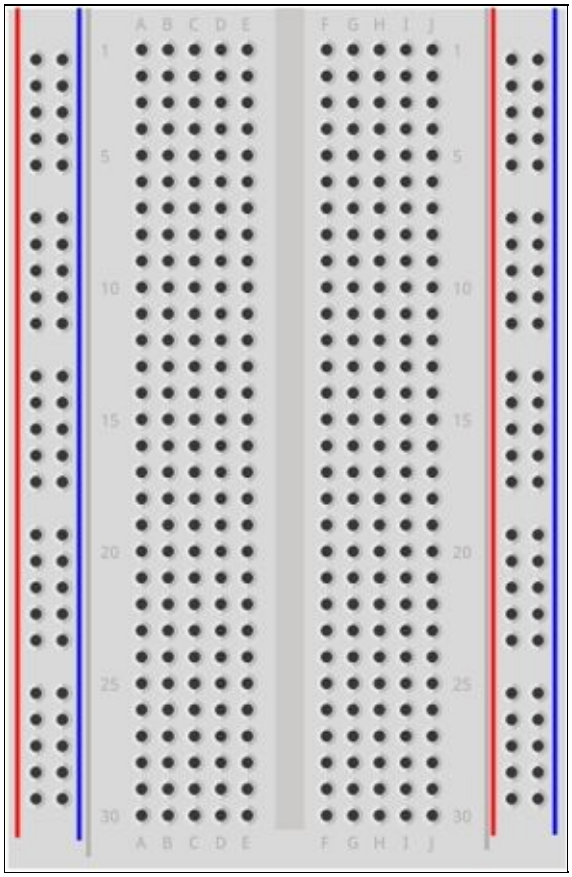


Figura 1.12 Utilizza cavi con l'anima rigida o dotati di connettori adeguati per collegare i componenti sulla breadboard.

> **NOTA** *Se imposti valori di tempo molto bassi rischi di vedere il LED restare sempre acceso. Non è un problema dello sketch o del compilatore: tutto sta funzionando a dovere, ma il LED si sta accendendo e spegnendo così velocemente (stiamo parlando di millisecondi) da non permettere all'occhio umano di percepire i due stati separatamente.*

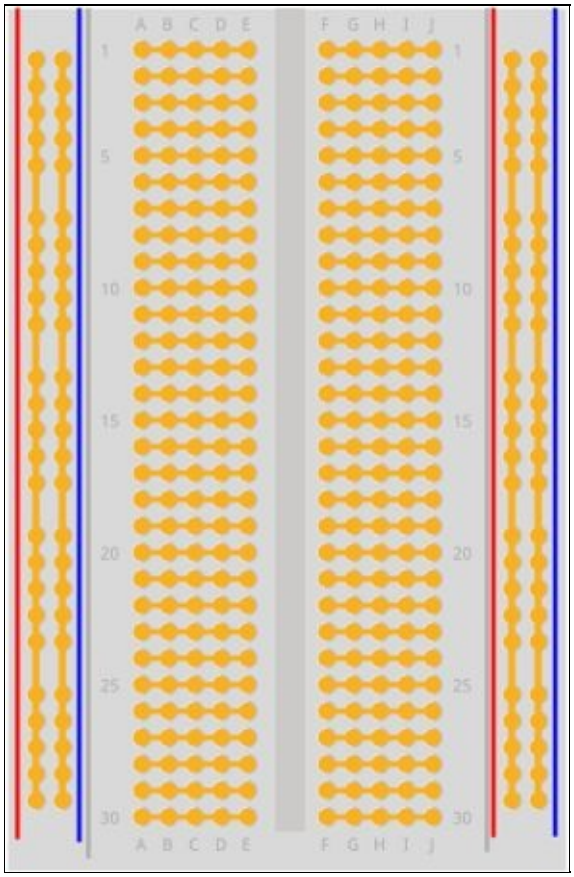
LA BREADBOARD

Uno strumento indispensabile per gli esperimenti con i componenti elettronici è la *breadboard*, in italiano chiamata anche *basetta sperimentale*. Si tratta di una base in plastica riutilizzabile con una griglia di fori nei quali si possono innestare i componenti e creare circuiti funzionanti, anche se provvisori. I componenti vengono mantenuti bloccati in posizione da piccole linguette metalliche a molla, quindi puoi spostare la breadboard con i componenti innestati senza paura di disfare il circuito. L'utilizzo di questo strumento offre il grande vantaggio di non dover eseguire saldature durante i test, permettendoti di modificare i circuiti con grande praticità e velocità. Quando hai terminato ti basta estrarre componenti e jumper (i cavi utilizzati per i collegamenti), senza correre il rischio di danneggiarli. La distanza tra i fori è di 2,54 mm, uno standard compatibile con i componenti elettronici DIP.



All’interno del guscio di plastica le breadboard nascondono piste di collegamento metalliche, disposte secondo una logica ben precisa: sui due lati più lunghi si trovano (quando presenti) le piste di alimentazione. I pin sono connessi tra loro in verticale, creando due linee utili per portare corrente ai componenti innestati nell’area principale.

Nell’area più interna i pin sono collegati tra loro in orizzontale, e il canale centrale vuoto suddivide ogni riga in due metà non connesse tra loro. Nella figura qui di seguito sono evidenziati in giallo, per chiarezza, i collegamenti standard dei pin.

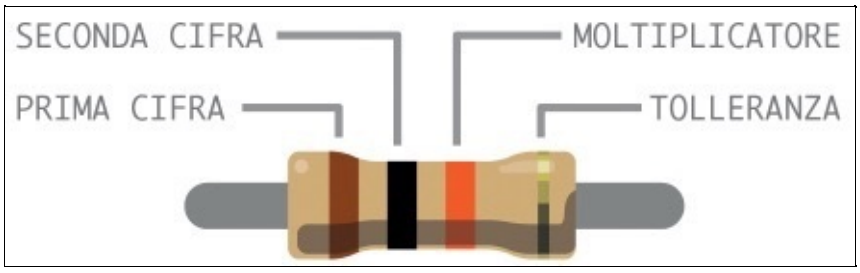


Ricorda che se hai dubbi puoi sempre utilizzare un tester o fare alcune prove con un LED per verificare i collegamenti delle piste nella tua breadboard.

Dopo questo chiarimento sull'uso della breadboard e sulle connessioni nascoste al suo interno hai tutti gli elementi per capire perché i componenti nella **Figura 1.11** sono stati disposti in quel modo per creare il circuito da abbinare allo sketch **Blink** modificato.

IL CODICE COLORE DEI RESISTORI

Puoi risalire al valore in Ohm dei resistori da utilizzare nei circuiti, per esempio quando sperimenti sulla breadboard, leggendo il codice colore impresso sul componente stesso. Ecco una tabella di riferimento per calcolarlo velocemente.



Colore	Prima cifra	Seconda cifra	Moltiplicatore	Tolleranza
Nero	–	0	1	
Marrone	1	1	10	1%
Rosso	2	2	100	2%
Arancione	3	3	1K	
Giallo	4	4	10K	
Verde	5	5	100K	0,50%

Blu	6	6	1M	0,25%
Viola	7	7		0,10%
Grigio	8	8		0,05%
Bianco	9	9		
Oro	–	–	0,1	5%
Argento	–	–	0,01	10%

Tieni presente che quando la banda relativa alla tolleranza non è presente, il valore indicato dalle altre tre bande ha un'approssimazione pari al 20%.

Potresti imbatterti anche in resistori con cinque o sei bande colorate. Nel caso di cinque bande, quella in più è da considerare prima del moltiplicatore, e rappresenta un'ulteriore cifra significativa. Nel caso di componenti a sei bande, l'ulteriore banda è da considerare a destra della tolleranza, ed è dedicata al coefficiente di temperatura: è un'indicazione che descrive di quanto varia la resistenza del componente al variare della temperatura a cui è sottoposto.

Se hai difficoltà a calcolare il valore dei resistori con la tabella, tramite una rapida ricerca su Internet puoi trovare degli strumenti dedicati proprio a questo: inserendo i colori che vedi sul componente ti viene subito restituito il suo valore.

Conclusione

In questo capitolo abbiamo illustrato la procedura di download dell'IDE Arduino e seguito il processo di installazione nei diversi sistemi operativi per cui il software è disponibile.

Abbiamo poi analizzato l'interfaccia dell'applicazione e imparato a riconoscere i componenti elettronici presenti sulla scheda Arduino Uno, prima di passare a un vero e proprio esperimento con lo sketch **Blink**.

Nel prossimo capitolo di dedicheremo a sketch e circuiti più complessi, per iniziare a conoscere in modo più approfondito il linguaggio di programmazione e capire come interagire con alcuni semplici componenti collegati alla scheda.

Input digitali e comunicazioni seriali

Nel capitolo precedente ci siamo concentrati sulla configurazione iniziale dell’ambiente di lavoro e sul trasferimento dello sketch Blink sulla scheda. Possiamo ora proseguire introducendo nuovi componenti, aumentando il grado di complessità degli sketch e descrivendo altre funzioni del linguaggio.

Lo sketch Button: elaborare input

Analizziamo un altro degli sketch di esempio: fai clic su **Apri** nella barra degli strumenti e scegli la voce **Button** dal menu **02.Digital**. Questo sketch, come descritto nel commento iniziale in inglese, permette ad Arduino di reagire alla pressione di un pulsante collegato a uno dei pin digitali della scheda.

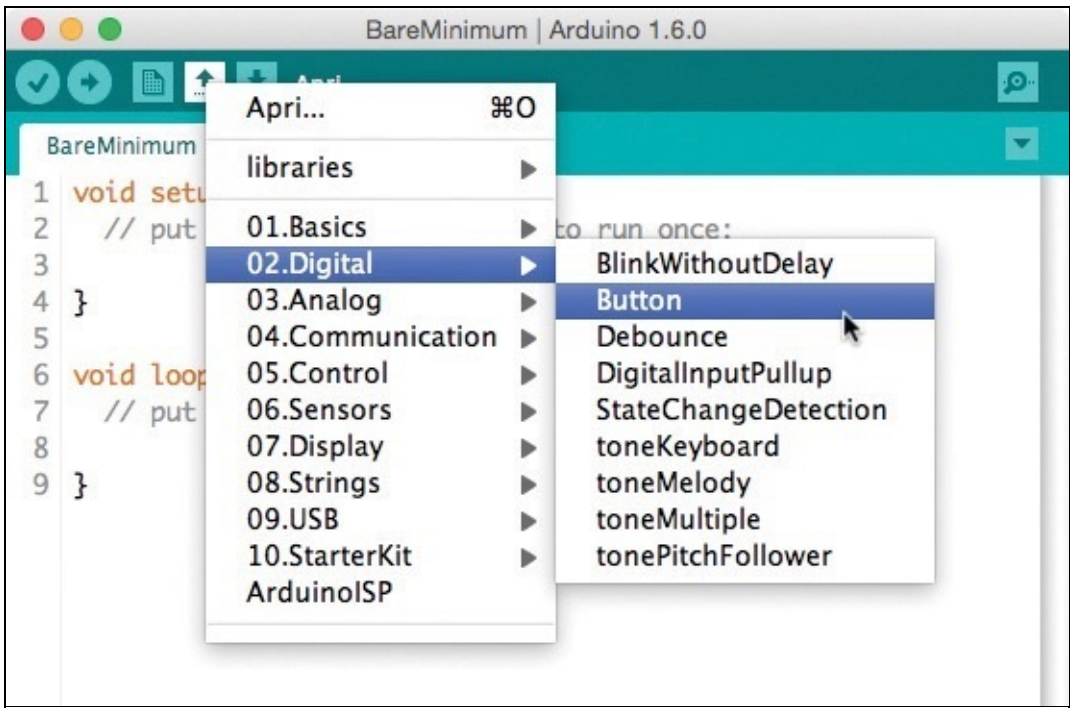


Figura 2.1 Carica lo sketch Button nell'IDE partendo dal menu Apri.

Nel commento iniziale ci sono già riferimenti ai componenti necessari e a come collegarli alla scheda:

```
The circuit:
* LED attached from pin 13 to ground
* pushbutton attached to pin 2 from +5V
* 10K resistor attached to pin 2 from ground
```

I componenti di cui abbiamo bisogno sono:

- un LED da collegare al pin `13` (possiamo sfruttare quello già integrato su Arduino Uno);
- un pulsante collegato tra il pin `2` e quello contrassegnato con `5V`;
- un resistore da 10k Ohm da collegare tra il pin `2` e `GND`.

Ecco nella figura una possibile disposizione sulla breadboard.

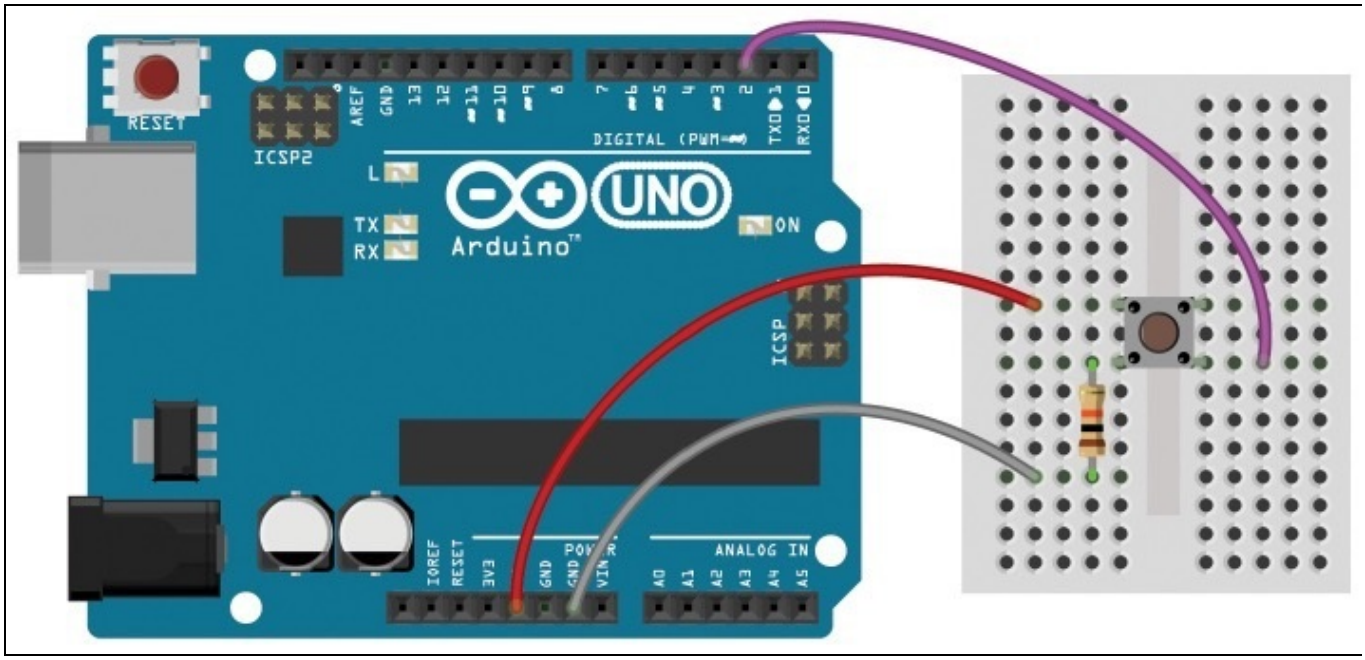


Figura 2.2 I componenti posizionati sulla breadboard e collegati tra loro.

> **NOTA** *Quando crei un circuito sulla breadboard cerca di mantenerlo per quanto possibile ordinato. Rispetta le convenzioni nella colorazione dei jumper: rosso per l'alimentazione (*5V*) e nero, marrone o grigio per* GND*.*

I pulsanti come quello nella figura sono normalmente utilizzati per realizzare circuiti stampati, e la distanza tra i loro pin li rende perfetti per essere posizionati sulla breadboard.

Questo pulsante è di tipo NO (*normally open*, ovvero normalmente aperto): quando è in posizione di riposo gli impulsi elettrici non possono attraversarlo, mentre premendolo permetti all'elettricità di scorrere.

> **NOTA** *Il pulsante ha quattro terminali, connessi tra loro a coppie. Posizionandolo a cavallo della fessura centrale noterai che puoi inserirlo solo in un verso. I due in alto nella* Figura 2.2 *sono sempre collegati tra loro, e lo stesso vale per i due in basso. Premendo il pulsante la corrente è libera di scorrere dall'alto verso il basso e viceversa. Quando il pulsante viene rilasciato il flusso è interrotto.*

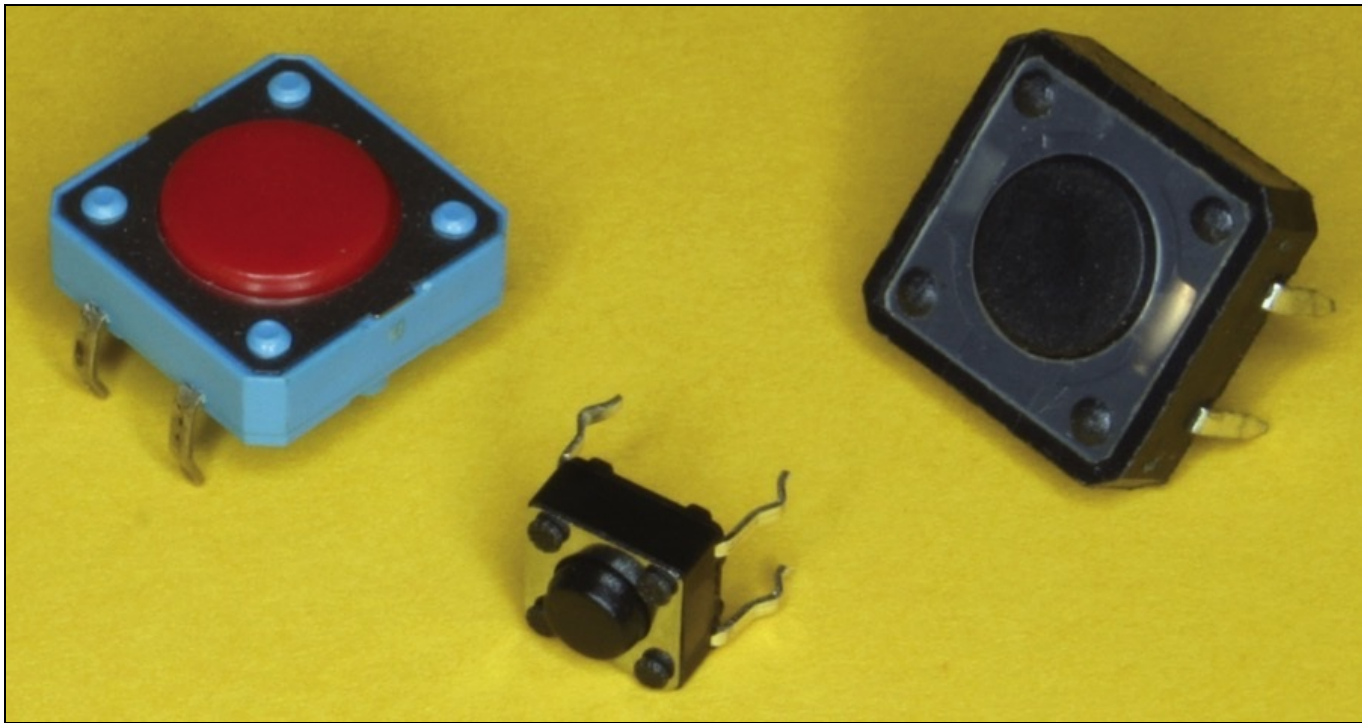


Figura 2.3 Esistono pulsanti di diverse dimensioni e caratteristiche. Quelli NO (normally open) permettono il passaggio di corrente solo quando il pulsante viene premuto. Quelli NC (normally close) si comportano in maniera opposta.

Prima di proseguire con l'analisi dello sketch cerchiamo di comprendere il comportamento del circuito che abbiamo realizzato sulla breadboard.

Il jumper rosso collega `5V` al terminale superiore del pulsante. Il jumper grigio, attraverso il resistore, collega `GND` all'altro terminale del componente. Il jumper viola, infine, connette il lato del pulsante già connesso a `GND` con il pin `2`.

Con il pulsante in posizione di riposo il pin `2` è connesso a `GND` (ricorda, i terminali sono collegati tra loro a coppie), e di conseguenza il voltaggio letto sarà di 0 Volt. Premendo il pulsante la corrente fornita dal jumper collegato a `5V` è libera di scorrere, portando il voltaggio letto dal pin `2` a 5 Volt.

In questo modo lo sketch in esecuzione sulla scheda è in grado di leggere dal pin `2` i valori `LOW` (che equivale a 0 Volt) e `HIGH` (5 Volt).

NOTA *Il resistore tra il pin* `GND` *e il pulsante impedisce, "consumando" corrente, il corto circuito tra* `5V` *e* `GND` *quando il pulsante viene premuto.*

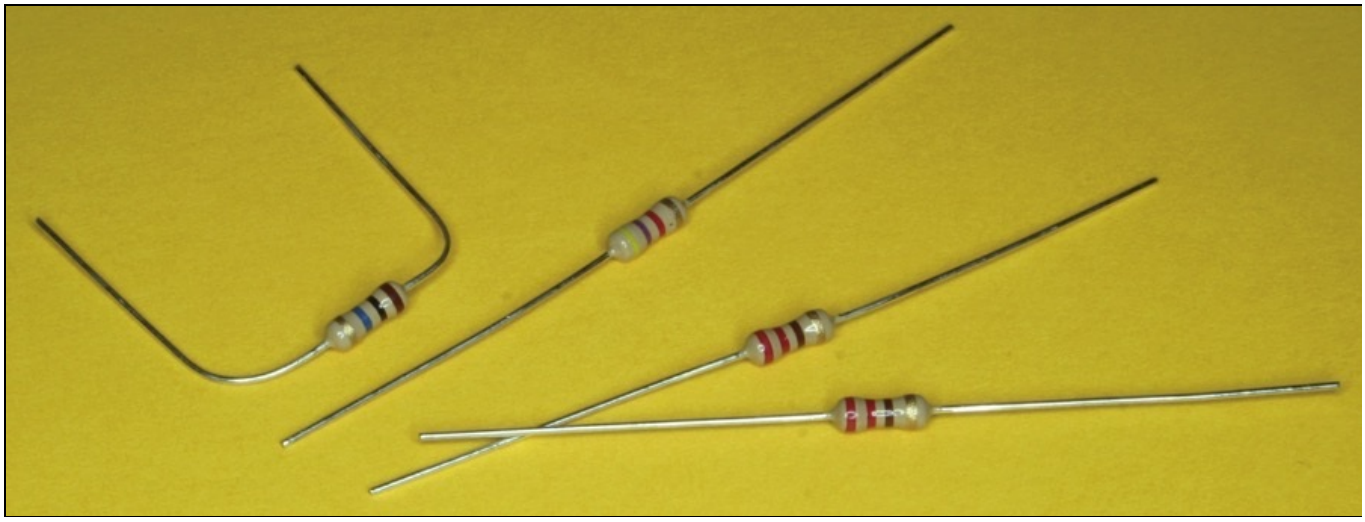


Figura 2.4 Puoi risalire al valore in Ohm di ogni resistore interpretandone le fasce colorate. Lo abbiamo visto alla fine del Capitolo 1.

Torniamo ora ad analizzare il codice: le prime istruzioni definiscono, come abbiamo visto nel capitolo precedente, le costanti `buttonPin` (per il pin a cui è collegato il pulsante) e `ledPin` (per il pin a cui è collegato il LED) che verranno utilizzate nello sketch:

```
const int buttonPin = 2;
// the number of the pushbutton pin

const int ledPin = 13;
// the number of the LED pin
```

L'istruzione successiva, analoga alle precedenti, serve per definire la variabile `buttonState`, il cui valore potrà essere modificato durante l'esecuzione dello sketch. Il tipo di dato attributio a questa variabile è `int`, e il valore iniziale è `0`.

La differenza rispetto alla dichiarazione delle costanti è l'assenza della parola chiave `const` all'inizio dell'istruzione:

```
int buttonState = 0;
// variable for reading the pushbutton status
```

A questo punto viene definita la funzione `setup()`, nella quale l'istruzione `pinMode()` si occupa di impostare i pin interessati come `OUTPUT` o `INPUT`:

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}
```

Come vedi non sono indicati direttamente i numeri dei pin coinvolti, ma in questa funzione e nella successiva si fa riferimento solo alle costanti definite all'inizio dello sketch.

Ora non resta che descrivere la funzione `loop()`, che conterrà le istruzioni da eseguire ciclicamente dopo l'avvio:

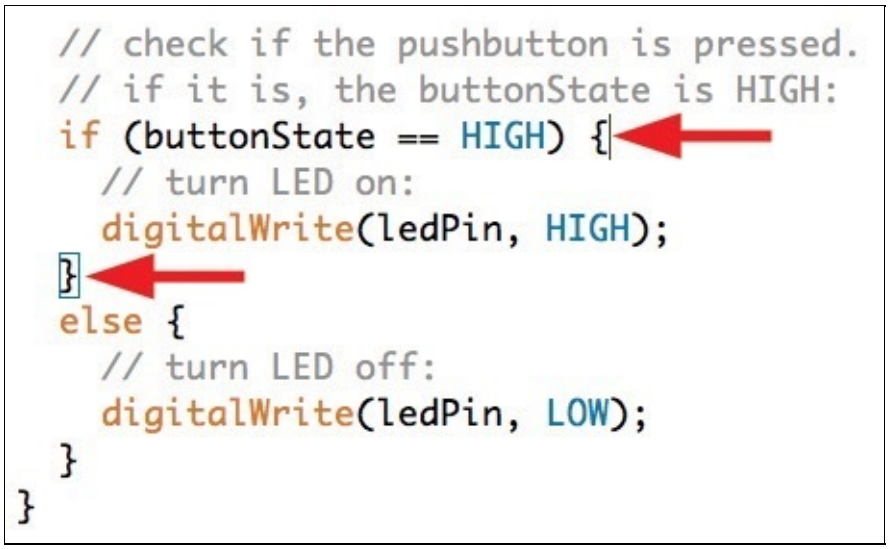
```
void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Con la prima istruzione la variabile `buttonState` assume valore `HIGH` o `LOW` in base al voltaggio del pin `2`, connesso al pulsante, e letto con `digitalRead()`. Analizzando il circuito abbiamo capito che in situazione di riposo il pin `2` è collegato a `GND`, perciò il valore letto sarà `LOW`.

Incontriamo poi un costrutto `if()`: se la condizione indicata tra parentesi è valida viene eseguito il blocco di istruzioni compreso tra le parentesi graffe `{` e `}`, altrimenti tale blocco viene saltato. In questo caso la verifica riguarda il valore assunto appena prima dalla variabile `buttonState`. Tra le parentesi graffe è contenuta una sola istruzione `digitalWrite()`, che imposta su `HIGH` il valore del pin a cui è collegato il LED, facendolo accendere.

LE COPPIE DI PARENTESI

Già con poche righe di codice lo sketch si allunga e la sua struttura inizia a complicarsi. L'IDE Arduino ti offre un piccolo strumento, a volte davvero utile, per orientarti nel codice: quando posizioni il cursore dopo una parentesi (graffa, tonda e persino quadra) nel codice viene evidenziata la "compagna" di apertura o chiusura. In questo modo puoi verificare con semplicità se la nidificazione delle istruzioni è corretta.



Nella figura le frecce indicano il cursore dopo la parentesi di apertura, e più in basso la parentesi di chiusura evidenziata automaticamente in azzurro.

Dopo la chiusura del blocco legato alla condizione `if` incontriamo la parola chiave `else`, in italiano “altrimenti”. In modo analogo permette di racchiudere tra parentesi graffe un blocco di istruzioni alternative: verranno eseguite solo quando la condizione indicata dopo `if` non si verifica. Nel nostro caso la funzione `digitalWrite()` imposterà su `LOW` il valore del pin al quale è connesso il LED, facendolo spegnere.

NOTA *Tieni presente che il blocco* `else` *non deve essere necessariamente presente dopo il blocco* `if`*. Quando c’è solo il blocco* `if` *e la condizione non si verifica, le istruzioni tra le parentesi graffe verranno semplicemente ignorate, e l’esecuzione del programma proseguirà con le istruzioni successive.*

Possiamo “leggere” questa sezione dello sketch in questo modo: “Se il pulsante è stato premuto fornisci corrente al pin del LED, altrimenti interrompila”.

GLI OPERATORI DI CONFRONTO

Quando utilizzi un costrutto `if` devi indicare tra parentesi una condizione da verificare. Questa condizione è un confronto matematico eseguito tra due valori. Ecco l’elenco dei possibili operatori:

- `a == b` (`a` è uguale a `b`);
- `a != b` (`a` è diverso da `b`);
- `a > b` (`a` è maggiore di `b`);
- `a < b` (`a` è minore di `b`);
- `a >= b` (`a` è maggiore o uguale a `b`);
- `a <= b` (`a` è minore o uguale a `b`);

Fai attenzione: la sintassi corretta del primo operatore, *uguale a*, è il simbolo `=` ripetuto due volte. Senza ripetizione il compilatore cercherà, come in una normale istruzione, di assegnare il valore scritto a destra dell’operatore all’elemento scritto a sinistra, generando un errore o un comportamento diverso da quello che ti aspetti.

Puoi confrontare una variabile con un valore (come per esempio `buttonState == HIGH` che troviamo nello sketch analizzato nelle pagine precedenti) oppure due variabili tra loro. Puoi anche eseguire operazioni matematiche sulle variabili.

Ricorda che per Arduino anche le variabili di tipo testuale hanno un equivalente numerico, perciò in casi particolari può tornarti utile eseguire verifiche anche su caratteri e stringhe.

Premi **Carica** nella barra degli strumenti dell’IDE per compilare lo sketch e trasferirlo sulla scheda. Quando il processo è completo (i LED `TX` e `RX` sulla scheda hanno smesso di illuminarsi), prova a premere il pulsante del circuito che hai preparato sulla breadboard: si illuminerà il LED `L` sulla scheda. Rilasciando il pulsante, il LED si spegnerà.

Comunicare da Arduino al computer

Finora ci siamo limitati a descrivere sketch che, una volta caricati su Arduino, funzionano in modo indipendente dal computer: la connessione USB viene utilizzata solo come fonte di alimentazione dopo il trasferimento iniziale dello sketch. La scheda Arduino, però, è anche in grado di comunicare tramite messaggi seriali durante l'esecuzione dello sketch. In questo modo può trasmettere o ricevere dati attraverso la connessione USB.

Il livello di interazione può diventare anche piuttosto complesso, grazie alla libertà di programmazione offerta dalla scheda e agli appassionati della community che arricchiscono e migliorano di giorno in giorno le librerie disponibili per interfacciarsi con diversi ambienti e linguaggi di programmazione.

Introduciamo alcune istruzioni in più nello sketch **Button** descritto nelle pagine precedenti, per far comunicare velocemente la scheda con il computer a cui è collegata.

Iniziamo aprendo la comunicazione seriale: per farlo abbiamo a disposizione l'istruzione `Serial.begin()` da utilizzare nella funzione `setup()`. Questa istruzione abilita la comunicazione. Come parametro prevede la velocità in *baud* (bit al secondo) della connessione con il computer. Utilizzeremo un valore standard di `9600`.

> **NOTA** *Il convertitore seriale-USB è connesso ai pin digitali* `0` *e* `1` *della scheda. Di conseguenza, se utilizzi la comunicazione seriale, evita di destinare tali pin al collegamento di altri componenti, per scongiurare comportamenti imprevisti e problemi di trasmissione.*

UN ACCORGIMENTO CON ARDUINO LEONARDO

Se stai utilizzando una scheda Arduino Leonardo o un'altra con processore ATmega32U4 (vedi l'**Appendice A**), dopo aver attivato la comunicazione seriale assicurati di verificare che la porta sia effettivamente attiva prima di proseguire, utilizzando questa sintassi:

```
Serial.begin(9600);
while(!Serial){;}
```

Ora che la connessione è attiva non ci resta che aggiungere nei punti giusti dello sketch l'istruzione `Serial.println()`. A ogni ciclo il messaggio passato alla funzione come parametro tra parentesi verrà trasmesso sulla porta seriale.

Per trasmettere una stringa di caratteri è necessario delimitarla con le virgolette doppie (`"`) all'interno della parentesi: `Serial.println("anno")` trasmetterà *anno*. Omettendo le virgolette, come in `Serial.println(anno)`, verrà invece trasmesso il valore assunto dall'ipotetica variabile `anno` definita nello sketch, e non la stringa *anno*.

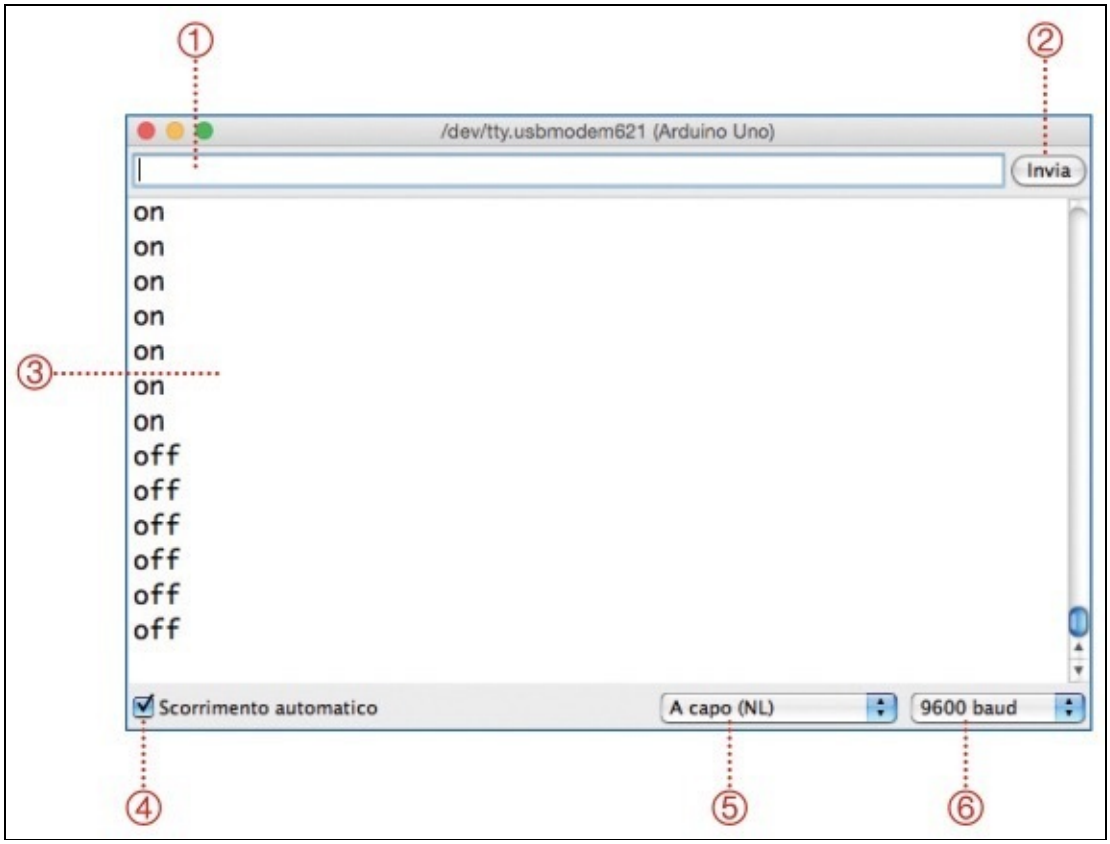
NOTA *Come vedi tutte le istruzioni riferite alla comunicazione seriale sono identificate dal prefisso* `Serial`.

Ecco qui di seguito il codice completo dello sketch **Button** modificato per trasmettere tramite porta seriale lo stato del LED (per chiarezza con i commenti tradotti in italiano).

```
/* Sketch Button modificato per comunicare
   lo stato del LED anche tramite porta seriale */
// dichiarazione delle costanti:
const int buttonPin = 2;
// il pin a cui è collegato il pulsante
const int ledPin = 13;
// il pin a cui è collegato il LED
// dichiarazione delle variabili:
int buttonState = 0;
// memorizza lo stato del pulsante
void setup() {
  // inizializzazione del pin del LED come uscita:
  pinMode(ledPin, OUTPUT);
  // inizializzazione del pin del pulsante come ingresso:
  pinMode(buttonPin, INPUT);
  // inizializzazione della comunicazione seriale:
  Serial.begin(9600);
}
void loop() {
  // lettura del pin a cui è connesso il pulsante:
  buttonState = digitalRead(buttonPin);
  // condizione if sullo stato del pulsante.
  // se premuto, buttonState assume valore HIGH:
  if (buttonState == HIGH) {
    // accensione del LED:
    digitalWrite(ledPin, HIGH);
    // trasmissione dello stato via porta seriale:
    Serial.println("on");
  }
  else {
    // spegnimento del LED:
    digitalWrite(ledPin, LOW);
    // trasmissione dello stato via porta seriale:
    Serial.println("off");
  }
}
```

Trasferisci lo sketch così modificato sulla scheda premendo **Carica** nell’IDE. Terminato il trasferimento verifica che le altre funzionalità dello sketch **Button** non siano state modificate: alla pressione del pulsante il LED L si comporta come prima.

L’unica differenza è il LED TX acceso sulla scheda: ti informa del traffico dati che sta avvenendo sulla porta seriale, e anche della sua direzione: TX monitora il traffico in uscita, mentre RX quello in entrata. Puoi visualizzare i messaggi trasmessi dalla scheda utilizzando uno degli strumenti integrati nell’IDE: fai clic sul pulsante **Monitor seriale** che trovi in alto a destra nella barra degli strumenti. Vedrai apparire la finestra mostrata nella figura qui di seguito, essenziale ma anche in questo caso dotata di tutte le funzionalità necessarie.



1. Il campo in cui digitare messaggi da trasmettere tramite la porta seriale. Più avanti nel capitolo vedremo come utilizzare questa possibilità.
2. Il pulsante **Invia** per confermare l'invio del messaggio digitato a fianco.
3. Lo spazio riservato ai messaggi in ingresso. Come vedi è già pieno di stringhe inviate dalla scheda.
4. Un segno di spunta su **Scorrimento automatico** consentirà alla finestra di visualizzare sempre i messaggi più recenti in basso, facendo fluire automaticamente quelli precedenti verso l'alto.
5. Il menu a tendina per scegliere la tipologia del carattere *EOL* (*end-of-line*) dei messaggi inviati verso la scheda.
6. Il menu a tendina per selezionare la velocità di connessione della comunicazione seriale. Impostalo sempre con un valore pari a quello che definisci nello sketch caricato sulla scheda, altrimenti avrai problemi di sincronizzazione durante la comunicazione in entrambe le direzioni.

Il monitor seriale si riempirà subito di righe con la stringa `off`. Come abbiamo detto, la funzione `loop()` viene eseguita a ciclo continuo.

Ora premi il pulsante sul circuito di prova che hai realizzato con la breadboard: il LED del pin `13` si illuminerà, e contemporaneamente il messaggio ricevuto dal monitor seriale diventerà `on` fino a quando non rilascerai il pulsante.

Con sole due istruzioni abbiamo abilitato lo sketch alla comunicazione seriale via USB.

IL DEBUG DEGLI SKETCH

Senza dubbio la comunicazione seriale è insostituibile per far interagire la scheda Arduino con il computer o con altri dispositivi. Durante lo sviluppo di nuovi sketch questa funzionalità può però tornare utile anche per individuare problemi e bug nel codice. Come hai visto nell'esempio appena descritto, far sì che uno sketch comunichi con il computer, trasferendo per esempio il valore assunto da una variabile o il risultato di un'operazione matematica, non richiede molte righe di codice. In alcuni casi poter leggere questi valori semplifica (e non poco) la risoluzione di problemi durante la realizzazione di un nuovo sketch.

Utilizza questo strumento nei tuoi sketch, e quando hai chiarito tutti i passaggi e sei pronto per la versione definitiva elimina tali istruzioni o, ancora meglio, commenta le relative righe per far sì che il compilatore le ignori ma restino a disposizione già pronte per futuri interventi sul codice.

Riconoscere la pressione del pulsante

Collegando il monitor seriale ci siamo resi conto della velocità con cui vengono eseguite le istruzioni nella funzione `loop()`: ogni riga con l'output `on` oppure `off` corrisponde a un ciclo dello sketch, perché a ogni ciclo lo sketch verifica se il pulsante è premuto o meno e lo comunica.

In molte situazioni è più utile identificare solo il cambiamento di stato del pulsante, ovvero l'esatto istante in cui viene premuto, per eseguire istruzioni legate a questo evento. Partendo dallo sketch dell'esempio precedente vediamo come "ripulire" l'output e individuare solo l'informazione che ci interessa.

> **NOTA** *Il passaggio di un segnale elettrico da* `LOW` *a* `HIGH` *è definito* rising edge*; quello contrario, da* `HIGH` *a* `LOW`*, è chiamato* falling edge*. Lo sketch è in grado di distinguere questi due eventi, che corrispondono alla pressione e al rilascio del pulsante: puoi attribuire un comportamento specifico a ognuno o ignorare uno dei due se necessario.*

Ecco per completezza l'intero sketch, con in evidenza le righe modificate che analizzeremo subito sotto:

```
/*
 Sketch Button modificato per comunicare
 solo il cambiamento di stato del pulsante
*/

// dichiarazione delle costanti:
const int buttonPin = 2;
// il pin a cui è collegato il pulsante
const int ledPin = 13;
// il pin a cui è collegato il LED

// dichiarazione delle variabili:
int buttonState = 0;
// memorizza lo stato del pulsante
int prevButtonState = LOW;
// memorizza lo stato precedente del pulsante

void setup() {
  // inizializzazione del pin del LED come uscita:
  pinMode(ledPin, OUTPUT);
  // inizializzazione del pin del pulsante come ingresso:
  pinMode(buttonPin, INPUT);
  // inizializzazione della comunicazione seriale:
  Serial.begin(9600);
}

void loop() {
  // lettura del pin a cui è connesso il pulsante:
  buttonState = digitalRead(buttonPin);
  // condizione if per riconoscere pressione e rilascio:
  if (buttonState != prevButtonState) {
    // condizione if sullo stato del pulsante.
    // se premuto, buttonState assume valore HIGH:
    if (buttonState == HIGH) {
      // accensione del LED:
      digitalWrite(ledPin, HIGH);
      // trasmissione dello stato via porta seriale:
      Serial.println("on");
    }
    else {
      // spegnimento del LED:
      digitalWrite(ledPin, LOW);
```

```
      // trasmissione dello stato via porta seriale:
      Serial.println("off");
    }
  }
  prevButtonState = buttonState;
}
```

Per prima cosa abbiamo bisogno di una variabile in più per memorizzare lo stato del pulsante letto nel ciclo precedente e confrontarlo con lo stato attuale. In questo modo potremo verificare se è cambiato. La dichiariamo insieme alle altre all'inizio dello sketch:

```
int prevButtonState = LOW;
// memorizza lo stato precedente del pulsante
```

Ora dobbiamo aggiungere una condizione `if` ed eseguire un ulteriore controllo: verifichiamo se lo stato del pin `2` è cambiato rispetto al ciclo precedente:

```
// condizione if per riconoscere pressione e rilascio:
if (buttonState != prevButtonState) {
```

Grazie a questa condizione, il codice contenuto tra le parentesi graffe viene eseguito solo quando il pulsante cambia stato: da `LOW` a `HIGH` o viceversa. Le condizioni `if` nidificate all'interno di questa distingueranno la pressione dal rilascio del pulsante.

Ricorda di aggiungere la parentesi graffa di chiusura del nuovo ciclo `if()` prima dell'ultima istruzione da aggiungere nel `loop()`, con la quale aggiorniamo il valore della variabile `prevButtonState`, attribuendole il valore letto con `buttonState` nel ciclo corrente. In questo modo sarà disponibile per eseguire il confronto nel ciclo successivo:

```
prevButtonState = buttonState;
```

A questo punto carica lo sketch aggiornato sulla scheda Arduino e apri il monitor seriale: appariranno nuovi messaggi `on` e `off` solo alla pressione e al rilascio del pulsante.

> **NOTA** *Puoi notare che il LED* `TX` *sulla scheda Arduino non è più acceso costantemente, ma lampeggia solo un istante quando premi o rilasci il pulsante: grazie alle modifiche effettuate, la comunicazione seriale è molto meno frequente rispetto a prima.*

Stesso input, comportamento differente

Riflettiamo su un aspetto importante della prototipazione con Arduino: abbiamo detto che tramite i pin possiamo collegare alla scheda sensori e attuatori. Con le istruzioni eseguite dal microcontrollore decidiamo come interpretare gli input e quali output eseguire. Ciò significa che questi sono due livelli della progettazione in simbiosi tra loro, ma separati: lo stesso circuito può avere comportamenti diversi in base alla logica definita nello sketch trasferito sulla scheda.

Ecco uno spunto per chiarire questo concetto: negli esempi precedenti il LED si illumina alla pressione del pulsante e si spegne al suo rilascio. Intervenendo solo sullo sketch, senza modificare il circuito, possiamo fare in modo che il LED si illumini a una prima pressione e resti acceso spegnendosi alla pressione successiva. Questo semplice e utile comportamento, come vedremo, nasconde delle complicazioni non banali.

Lo sketch precedente, grazie alle funzioni `if` nidificate, è in grado di distinguere la pressione (*rising edge*) e il rilascio (*falling edge*) del pulsante.

Per fare in modo che il LED resti acceso al rilascio del pulsante reagiremo al solo *rising edge*. La sintassi che useremo per concatenare due verifiche in un'unica condizione `if` è la seguente:

```
if (buttonState != prevButtonState && buttonState == HIGH){
```

Tra le due condizioni è evidenziato il simbolo `&&`, l'operatore che equivale a un *and* logico. Le istruzioni contenute nel ciclo `if` saranno eseguite solo se lo stato del pulsante è cambiato (prima condizione) e il nuovo stato del pulsante è `HIGH` (seconda condizione).

GLI OPERATORI BOOLEANI

`&&` è solo uno degli operatori utilizzati per concatenare più condizioni. Esistono gli equivalenti di *e* (in inglese *and*), *o* (in inglese *or*) e *non* (in inglese *not*). Ecco qui di seguito la relativa sintassi, che può tornarti utile all'interno di una dichiarazione condizionale `if`:

- `&&` (*and*): restituisce `true` se tutte le condizioni sono valide;
- `||` (*or*): restituisce `true` se almeno una delle condizioni è valida;
- `!` (*not*): restituisce `true` se la condizione seguente non è valida.

Vediamo lo sketch completo con evidenziate le modifiche effettuate, per poi analizzarle nel dettaglio:

```
/*
 Sketch Button modificato per reagire
 solo al rising edge del pulsante
```

```
*/

// dichiarazione delle costanti:
const int buttonPin = 2;
// il pin a cui è collegato il pulsante
const int ledPin = 13;
// il pin a cui è collegato il LED

// dichiarazione delle variabili:
int buttonState = 0;
// memorizza lo stato del pulsante
int prevButtonState = LOW;
// memorizza lo stato precedente del pulsante
int ledState = LOW;
// memorizza lo stato del LED

void setup() {
  // inizializzazione del pin del LED come uscita:
  pinMode(ledPin, OUTPUT);
  // inizializzazione del pin del pulsante come ingresso:
  pinMode(buttonPin, INPUT);
  // inizializzazione della comunicazione seriale:
  Serial.begin(9600);
}

void loop() {
  // lettura del pin a cui è connesso il pulsante:
  buttonState = digitalRead(buttonPin);
  // condizione if combinata:
  if (buttonState != prevButtonState && buttonState == HIGH) {
    // inversione dello stato del LED
    ledState = !ledState;
    if (ledState == HIGH) {
      digitalWrite(ledPin, HIGH);
      Serial.println("on");
    }
    else {
      digitalWrite(ledPin, LOW);
      Serial.println("off");
    }
  }
  prevButtonState = buttonState;
}
```

Nella prima sezione dello sketch dichiariamo un'ulteriore variabile per memorizzare lo stato del LED. Questo è inizialmente spento, perciò il valore iniziale che attribuiamo alla variabile è `LOW`:

```
int ledState = LOW;
// memorizza lo stato del LED
```

All'interno del ciclo `if()` la prima istruzione modifica il valore di `ledState`:

```
ledState = !ledState;
```

Il simbolo `!` (operatore *not*) prima del nome della variabile ne inverte il valore: da `LOW` diventerà `HIGH` e viceversa, da `HIGH` diventerà `LOW`.

In base al valore di `ledState`, il ciclo `if()` nelle righe seguenti accende o spegne il LED e ne comunica lo stato tramite la connessione seriale.

ABBREVIAZIONI E MALIZIE

La versatilità del linguaggio di programmazione Arduino, ereditata da C, permette di poter ottenere lo stesso risultato utilizzando sintassi differenti. Per esempio, il ciclo `if()` all'interno di quello principale

potrebbe essere riassunto in sole due righe di codice. La prima istruzione sarebbe necessaria per attribuire al pin del LED lo stesso valore di `ledState`, quindi `HIGH` o `LOW`:

```
digitalWrite(ledPin, ledState);
```

La seconda istruzione, definita operatore ternario, è la sintassi abbreviata di un ciclo `if()` semplice. Prima del simbolo `?` viene dichiarata la condizione, in questo caso `ledState == HIGH`. Subito dopo ecco l'struzione da eseguire se la condizione è soddisfatta, ovvero la comunicazione seriale *on*. Il simbolo `:` separa questa istruzione dalla seguente (la comunicazione seriale *off*), da eseguire se la condizione non è soddisfatta:

```
ledState == HIGH ? Serial.println("on") : Serial.println("off");
```

Esistono molti stili di programmazione, alcuni più efficaci di altri, alcuni meno immediati da leggere e interpretare di altri. In generale, cerca di prediligere sempre l'ordine, la chiarezza e la comprensibilità dei tuoi sketch, ricordando di aggiungere commenti dove necessario.

Nell'IDE Arduino verifica il codice e trasferiscilo sulla scheda con il pulsante **Carica**. Quando appare il messaggio **Caricamento completato** nella barra di notifica, apri la finestra del monitor seriale. Prova ora a premere il pulsante sulla breadboard: il LED `13` si accenderà sulla scheda, e nel monitor seriale apparirà il messaggio **on**. Ora rilascia il pulsante: il LED resterà acceso e non appariranno nuovi messaggi seriali. Premi di nuovo il pulsante sulla breadboard: il LED `13` si spegnerà, e nel monitor seriale apparirà il messaggio **off**.

Attenzione! Facendo qualche tentativo con il pulsante ci accorgiamo che a volte, in modo casuale, a una pressione del pulsante corrispondono più operazioni *on* e *off* registrate.

Qual è la ragione di questo fenomeno? La logica utilizzata nello sketch è impeccabile, e sappiamo per certo che il microcontrollore sulla scheda non ha problemi a eseguirla in modo sempre perfetto. Dobbiamo ricercare la causa nella parte fisica del prototipo: il pulsante.

Una complicazione: il debounce di un pulsante

Realizzando lo sketch precedente ci siamo imbattuti in una complicazione non banale. A volte il comportamento del prototipo a cui stiamo lavorando non è quello che ci aspetteremmo: il LED si accende o si spegne anche quando non dovrebbe. A cosa è dovuta questa anomalia? Pensiamo un attimo a come è costruito il pulsante che premiamo sulla breadboard: al suo interno una sottile piastra metallica, modellata seguendo una forma ben precisa, scatta in posizione di contatto quando esercitiamo pressione, e torna elasticamente nella posizione di riposo quando rimuoviamo il dito. È proprio questa elasticità a creare falsi contatti e a far interpretare a volte in modo errato allo sketch quello che sta accadendo.

In inglese questo comportamento è chiamato *bounce* (rimbalzo). Come vedi nello schema della **Figura 2.5**, se i picchi dei "rimbalzi" sono abbastanza forti, possono essere interpretati dal microcontrollore come nuovi momenti di `HIGH` e `LOW`, ovvero come ulteriori pressioni del pulsante, anche se durano solo pochi millisecondi.

Figura 2.5 Uno schema molto semplificato del fenomeno bounce.

Questo fenomeno è legato allo costruzione del componente. La sua intensità e le caratteristiche variano a seconda del modello, dei materiali, dei processi produttivi, dell'usura e così via. Alcuni pulsanti potrebbero risultare più sensibili, altri meno.

Possiamo ridurre le possibilità di registrare falsi segnali con tecniche di debounce: ritardiamo per un brevissimo intervallo di tempo l'esecuzione del codice legato alla pressione del pulsante, o con più controlli sullo stato del pin assicuriamoci di reagire solo a una vera pressione. Modifichiamo lo sketch che abbiamo costruito negli esempi precedenti integrando una tecnica semplice e piuttosto efficace.

Nella prima parte dello sketch aggiungiamo una costante, per mantenere ordine e leggibilità. Inseriamo poi un leggero ritardo con l'istruzione `delay()` (già utilizzata

nello sketch **Blink**) nell'esecuzione dello sketch: daremo modo al pulsante di assestarsi prima del ciclo successivo:

```
/* Sketch Button modificato con debounce */

// dichiarazione delle costanti:
const int buttonPin = 2;
// il pin a cui è collegato il pulsante
const int ledPin = 13;
// il pin a cui è collegato il LED
const int debounceDelay = 50;
// l'assestamento in millisecondi
// dichiarazione delle variabili:
int buttonState = 0;
// memorizza lo stato del pulsante
int prevButtonState = LOW;
// memorizza lo stato precedente del pulsante
int ledState = LOW;
// memorizza lo stato del LED

void setup() {
  // inizializzazione del pin del LED come uscita:
  pinMode(ledPin, OUTPUT);
  // inizializzazione del pin del pulsante come ingresso:
  pinMode(buttonPin, INPUT);
  // inizializzazione della comunicazione seriale:
  Serial.begin(9600);
}

void loop() {
  // lettura del pin a cui è connesso il pulsante:
  buttonState = digitalRead(buttonPin);
  // condizione if combinata:
  if (buttonState != prevButtonState && buttonState == HIGH) {
    // inversione dello stato del LED
    ledState = !ledState;
    if (ledState == HIGH) {
      delay(debounceDelay);
      digitalWrite(ledPin, HIGH);
      Serial.println("on");
    }
    else {
      digitalWrite(ledPin, LOW);
      Serial.println("off");
    }
  }
  prevButtonState = buttonState;
}
```

Compila lo sketch e caricalo sulla scheda: grazie a questo piccolo accorgimento il riconoscimento della pressione del pulsante è molto più affidabile. Puoi fare delle prove modificando il valore di `debounceDelay` e adattare al meglio lo sketch alle caratteristiche meccaniche del pulsante che stai utilizzando.

NOTA *Con questa semplice tecnica viene necessariamente introdotto un ritardo (di pochi millisecondi, quasi impercettibile) nell'esecuzione delle istruzioni. Esistono anche altre tecniche per filtrare gli input affetti da questo tipo di problemi, che non comportano l'utilizzo dell'istruzione* `delay()` *e che dunque non ritardano l'esecuzione di eventuali altre funzioni. Trovi un esempio completo nello sketch* 02.Digital > Debounce*.*

Comunicare dal computer verso Arduino

Fino a questo momento abbiamo utilizzato la comunicazione seriale solo in una direzione: dalla scheda Arduino verso il computer. Facciamo ora un ulteriore passo in avanti: instauriamo una comunicazione bidirezionale, facendo in modo che la scheda reagisca ai comandi inviati dal computer.

In questo esempio non utilizzeremo la breadboard, sarà sufficiente la scheda Arduino: invieremo le informazioni in input tramite la connessione USB e agiremo sul pin `13` per comandare il LED `L` integrato.

> **NOTA** *La comunicazione dal computer è immune dal problema legato al debounce che abbiamo analizzato poco fa: i messaggi sono digitali, e perciò arrivano alla scheda già filtrati e ripuliti.*

Creiamo e analizziamo lo sketch suddividendolo in blocchi. Partiamo dalla dichiarazione di costanti e variabili:

```
// dichiarazione delle costanti:
const int ledPin = 13;
// il pin a cui è collegato il LED

// dichiarazione delle variabili:
char tastoPremuto;
// memorizza il tasto premuto sulla tastiera
```

Come sempre dichiariamo per comodità le costanti relative ai pin che utilizzeremo, per poter adattare velocemente lo sketch. In questo caso utilizzeremo solo il LED collegato al pin `13`. Segue la dichiarazione delle variabili: in questo sketch definiamo la variabile `tastoPremuto`, per memorizzare le informazioni ricevute tramite la connessione seriale.

Passiamo a questo punto alla funzione `setup()`:

```
void setup() {
  // il pin del LED è impostato come uscita:
  pinMode(ledPin, OUTPUT);
  // inizializzazione della comunicazione seriale:
  Serial.begin(9600);
  Serial.println("In attesa di comandi");
}
```

Utilizziamo tre istruzioni che abbiamo già incontrato: `pinMode()` per impostare il pin `13` come `OUTPUT`, `Serial.begin()` per attivare la comunicazione seriale e `Serial.println()` per fornire un feedback all'utente. Non resta che completare con la funzione `loop()`:

```
void loop() {
  if (Serial.available()) {
    tastoPremuto = Serial.read();
    switch (tastoPremuto) {
      case 72:
        // la codifica ASCII del carattere H è 72
        digitalWrite(ledPin, HIGH);
        Serial.println("LED on");
        break;
```

```
      case 76:
        // la codifica ASCII del carattere L è 76
        digitalWrite(ledPin, LOW);
        Serial.println("LED off");
        break;
      case 13:
        // la codifica ASCII di "A capo" è 13
        // È da ignorare: nessun comando
        break;
      case 10:
        // la codifica ASCII di "Nuova riga" è 10
        // È da ignorare: nessun comando
        break;
      default:
        // le istruzioni di default sono eseguite
        // se nessuna delle precedenti condizioni
        // si è verificata
        Serial.print("Comando sconosciuto: ");
        Serial.print(tastoPremuto);
        Serial.println(" (premi H o L)");
    }
  }
}
```

La prima istruzione è una condizione `if()`: il blocco di codice verrà eseguito solo quando `Serial.available()` restituisce valore `TRUE`, ovvero se sono stati ricevuti dati tramite la porta seriale.

All'interno della condizione troviamo subito l'istruzione `tastoPremuto = Serial.read()`: il contenuto del buffer di ricezione seriale viene trasferito alla variabile `tastoPremuto`.

A questo punto dobbiamo fare in modo che lo sketch reagisca in modo diverso a seconda dell'input ricevuto: abbiamo bisogno di definire un comportamento per la ricezione di `H` (che utilizziamo come abbreviazione di *High*) e di `L` (abbreviazione di *Low*). Ci preoccupiamo anche di fornire un messaggio di errore nel caso venga ricevuto un valore diverso da quelli che abbiamo deciso di utilizzare nello sketch.

> **NOTA** *È sempre consigliabile prevedere l'invio di feedback all'utente per descrivere eventuali errori o situazioni anomale, anche banali. In questo caso lo sketch gestisce solo messaggi in ingresso* `H` *o* `L`*. Negli altri casi il messaggio di errore informa l'utente e previene ulteriori errori.*

Definiamo la reazione ai diversi caratteri ricevuti con una routine `switch()`, valida alternativa all'uso di più cicli `if()` quando la variabile da verificare può assumere solo un certo numero di valori predefiniti. La sintassi prevede la dichiarazione di una variabile tra parentesi tonde e un certo numero di condizioni `case`, una per ogni valore possibile. Rivediamo un dettaglio del codice:

```
switch (tastoPremuto) {
  case 72:
    // la codifica ASCII del carattere H è 72
    digitalWrite(ledPin, HIGH);
    Serial.println("LED on");
    break;
```

Quando `tastoPremuto` ha valore `72` (l'equivalente del carattere `H` nella tabella ASCII), le righe di codice fino all'istruzione `break` vengono eseguite, altrimenti vengono

saltate per passare alla verifica successiva.

Il codice all'interno della condizione `default` viene eseguito quando nessuna delle condizioni precedenti si verifica. È l'equivalente del costrutto `else` abbinato a `if()`.

Con questa sintassi è molto semplice definire in modo ordinato comportamenti diversi per i possibili valori di una variabile: prova a immaginare le istruzioni necessarie per ottenere lo stesso comportamento con una serie di cicli `if()` e un `else` finale.

NOTA *Trovi la tabella di riferimento dei valori ASCII all'indirizzo* `http://www.arduino.cc/en/Reference/ASCIIchart`. *I valori vanno da 0 a 127, e comprendono lettere, numeri, simboli e caratteri non stampabili.*

Un'ultima precisazione: nel costrutto `case` descritto vengono presi in considerazione anche due valori ASCII speciali: `13` e `10`. Ci stiamo preoccupando di intercettare i caratteri di termine riga inviati dal monitor seriale dopo ogni istruzione per ignorarli, non eseguendo alcuna azione.

In alternativa puoi impedire che il monitor seriale invii questi caratteri modificando l'impostazione dal relativo menu a tendina in basso nell'interfaccia (**Figura 2.6**).

Ora lo sketch è completo: esegui l'ormai familiare procedura di verifica e trasferimento del codice sulla scheda. Quando i LED `TX` e `RX` si spengono e nell'IDE Arduino leggi il messaggio **Caricamento completato**, apri la finestra del monitor seriale, questa volta non solo per leggere messaggi ma anche per inviarne.

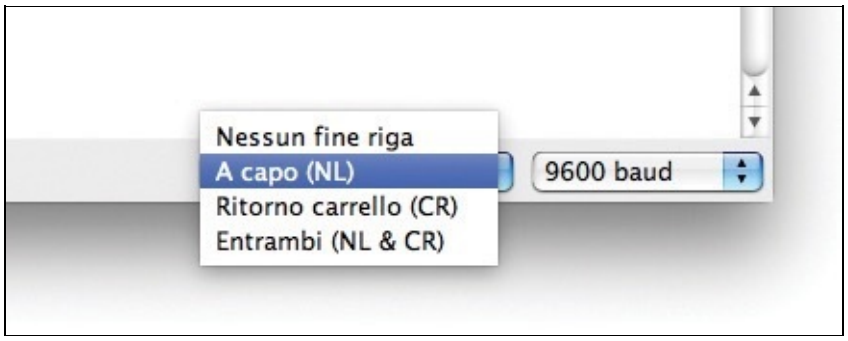


Figura 2.6 Nella parte bassa del monitor seriale puoi decidere come chiudere i messaggi inviati alla scheda.

Digita **H** e premi Invio sulla tastiera: la scheda riconoscerà il messaggio, rispondendo con la stringa **LED on** nel monitor seriale e attivando il LED collegato al pin `13`.

NOTA *Fai attenzione: nella tabella ASCII i caratteri sono* case sensitive*. Ciò significa che lettere minuscole e maiuscole sono identificate da codici differenti:* `h` *è diverso da* `H`*. Prova a integrare*

nello sketch anche il riconoscimento di questi caratteri!

Invia adesso il carattere L: la scheda risponderà con **LED off** e il LED del pin `13` si spegnerà. Prova a inviare qualsiasi altro carattere: la parte di ciclo `case()` eseguita sarà quella di default. Il risultato sarà il messaggio seriale **Comando sconosciuto: X (Premi H o L)** e lo stato del LED non cambierà.

Conclusione

In questo capitolo abbiamo imparato a conoscere a fondo due funzionalità fondamentali, che ritroverai in molti sketch.

La prima è la gestione degli input digitali, indispensabile per interagire con quei sensori che forniscono una lettura digitale di un fenomeno (negli esempi abbiamo visto lavorare con un pulsante). Per “lettura digitale” ricorda che ci riferiamo a una lettura del tipo acceso/spento (nel linguaggio “elettrico” `HIGH`/`LOW`), contrapposta alla lettura di fenomeni analogici che analizzeremo nel prossimo capitolo.

La seconda è l’utilizzo dell’interfaccia di comunicazione seriale della scheda, indispensabile per il debug e base di ogni interazione con altri componenti “intelligenti”, come per esempio software in esecuzione sul computer o altre schede elettroniche evolute.

Input e output analogici

Non tutti i fenomeni elettrici possono essere ricondotti ai solo stati HIGH e LOW, acceso o spento. Come può una scheda elettronica essere in grado di interagire con valori diversi dagli 1 e 0 dei bit?

Gli output analogici: i pin PWM

Nei capitoli precedenti abbiamo visto come possiamo intervenire sullo stato dei pin con l'istruzione `digitalWrite()`, scegliendo tra `HIGH` e `LOW`. Il microcontrollore della scheda Arduino Uno non è in grado di attribuire valori intermedi, ma la velocità di esecuzione del codice è tale da permetterci di utilizzare una tecnica definita *PWM* (*Pulse-width modulation*, in italiano modulazione della larghezza di impulso) per approssimare i valori e gestire esigenze del genere.

Con questa tecnica il pin passa continuamente dallo stato `HIGH` allo stato `LOW`, e viceversa, con una frequenza di molti cicli al secondo. Il rapporto tra tempo passato in stato `HIGH` e tempo in stato `LOW` identifica il valore medio del segnale trasmesso. Possiamo così approssimare un segnale analogico utilizzando solo i mezzi digitali che abbiamo a disposizione.

NOTA *Alcune schede come la Arduino Due, più evoluta, dispongono di circuiti DAC, ovvero di convertitori digitale-analogico integrati. Utilizzando queste schede è possibile generare segnali realmente analogici, con forme d'onda differenti da quella descritta qui di seguito.*

L'istruzione che utilizziamo per gestire questa funzionalità è `analogWrite()`, che richiede la dichiarazione di due parametri:

- il numero del pin a cui ci stiamo riferendo: tieni presente che solo i pin contrassegnati sulla scheda con il simbolo `~` hanno funzionalità PWM (sulla scheda Arduino Uno sono i pin digitali `3`, `5`, `6`, `9`, `10` e `11`);
- il valore che definisce il rapporto tra la durata dei segnali `HIGH` e `LOW`. Questo parametro ha una risoluzione di 8 bit, e può assumere valori numerici compresi tra `0`, pari a un pin sempre in stato `LOW`, e `255`, pari a un pin sempre in stato `HIGH`.

NOTA *Ricordi lo sketch* Blink *del* Capitolo 1*? Impostando valori di* `delay()` *molto bassi il nostro occhio non è in grado di percepire i singoli stati* `HIGH` *e* `LOW`*, e il LED ci appare solo meno illuminato. Poiché in quello sketch gli intervalli di accensione e spegnimento sono uguali tra loro, è quasi come se stessimo definendo un comportamento PWM al 50%.*

Per fare un esempio, `analogWrite(9, 63)` attiva il pin `9` al 25%. Ecco qui di seguito una visualizzazione di alcuni rapporti tra `HIGH` e `LOW`, per comprendere meglio il fenomeno.

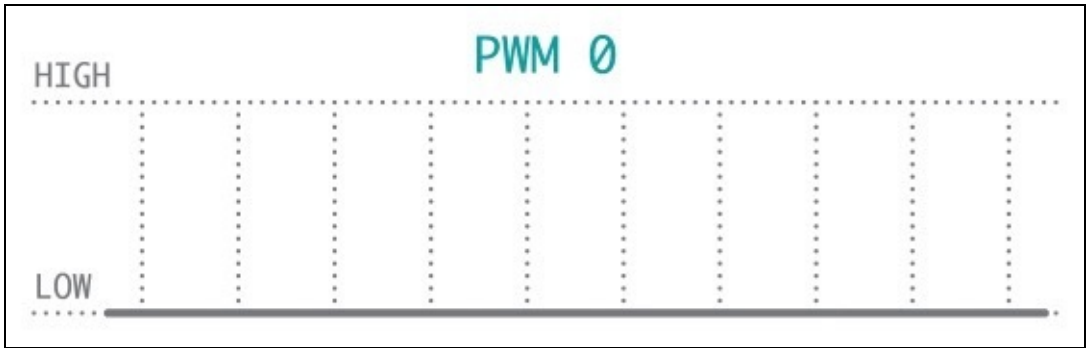


Figura 3.1 L’istruzione analogWrite(pin, 0) genera un segnale PWM allo 0%.

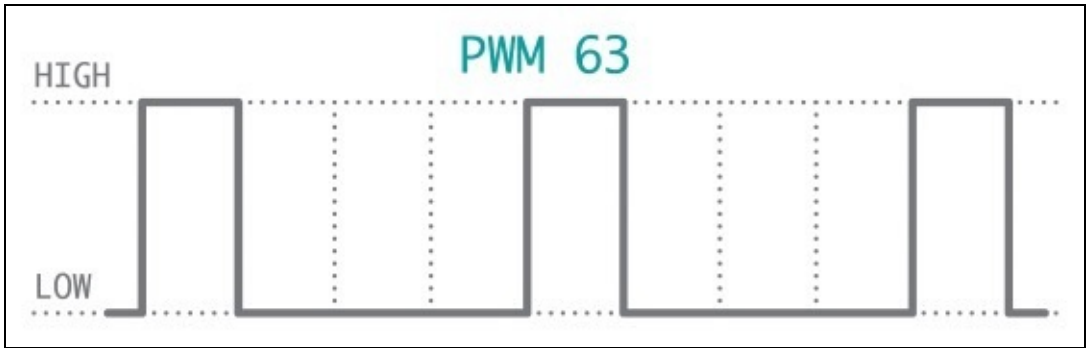


Figura 3.2 L’istruzione analogWrite(pin, 63) genera un segnale PWM al 25%.

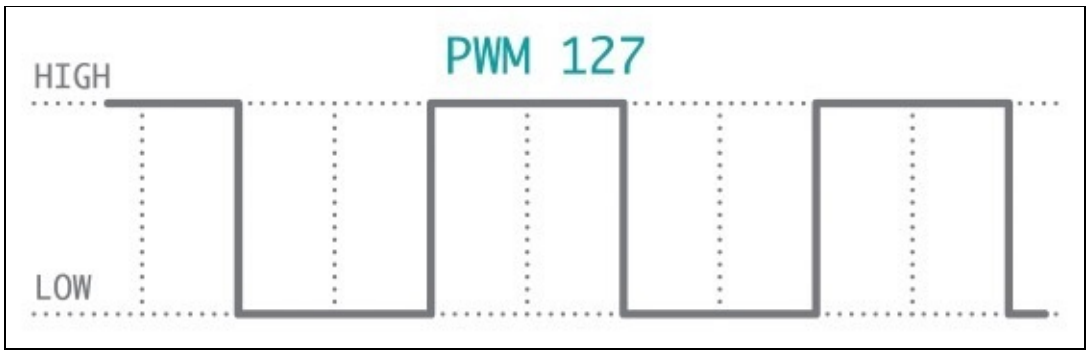


Figura 3.3 L’istruzione analogWrite(pin, 127) genera un segnale PWM al 50%.

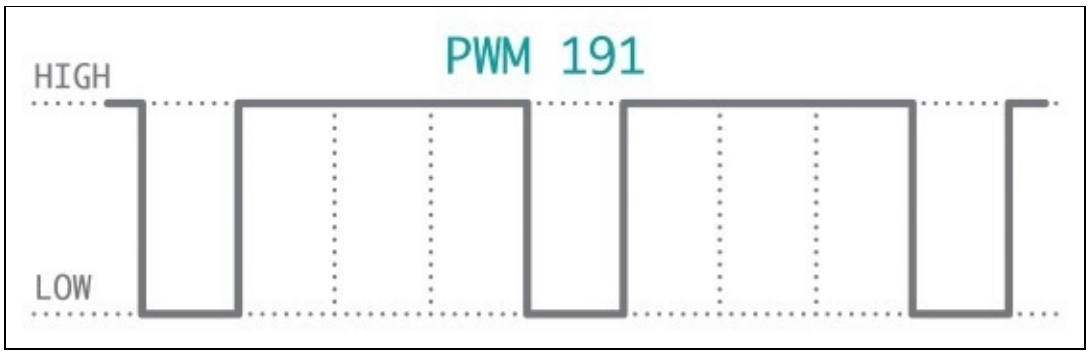


Figura 3.4 L’istruzione analogWrite(pin, 191) genera un segnale PWM al 75%.

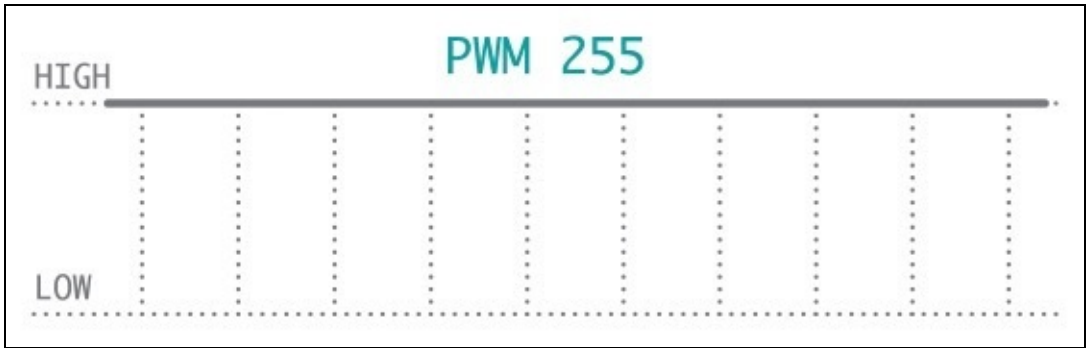


Figura 3.5 L’istruzione analogWrite(pin, 255) genera un segnale PWM al 100%.

Lo sketch Fading: la tecnica PWM in azione

Analizziamo uno sketch che sfrutta questa possibilità di parzializzare l'output di un pin per simulare un valore analogico.

Nell'IDE Arduino fai clic su **Apri** nella barra degli strumenti, e tra gli esempi scegli **03.Analog** > **Fading**.

Nel commento iniziale, oltre ai crediti, troviamo la descrizione del semplice circuito da realizzare per testare lo sketch:

```
The circuit:
* LED attached from digital pin 9 to ground.
```

Innesta il LED sulla breadboard, collega in serie un resistore da 220 Ohm e connetti ai pin `9` e `GND`. Ricorda di fare attenzione alla polarità del LED, poiché la corrente può attraversarlo solo in una direzione.

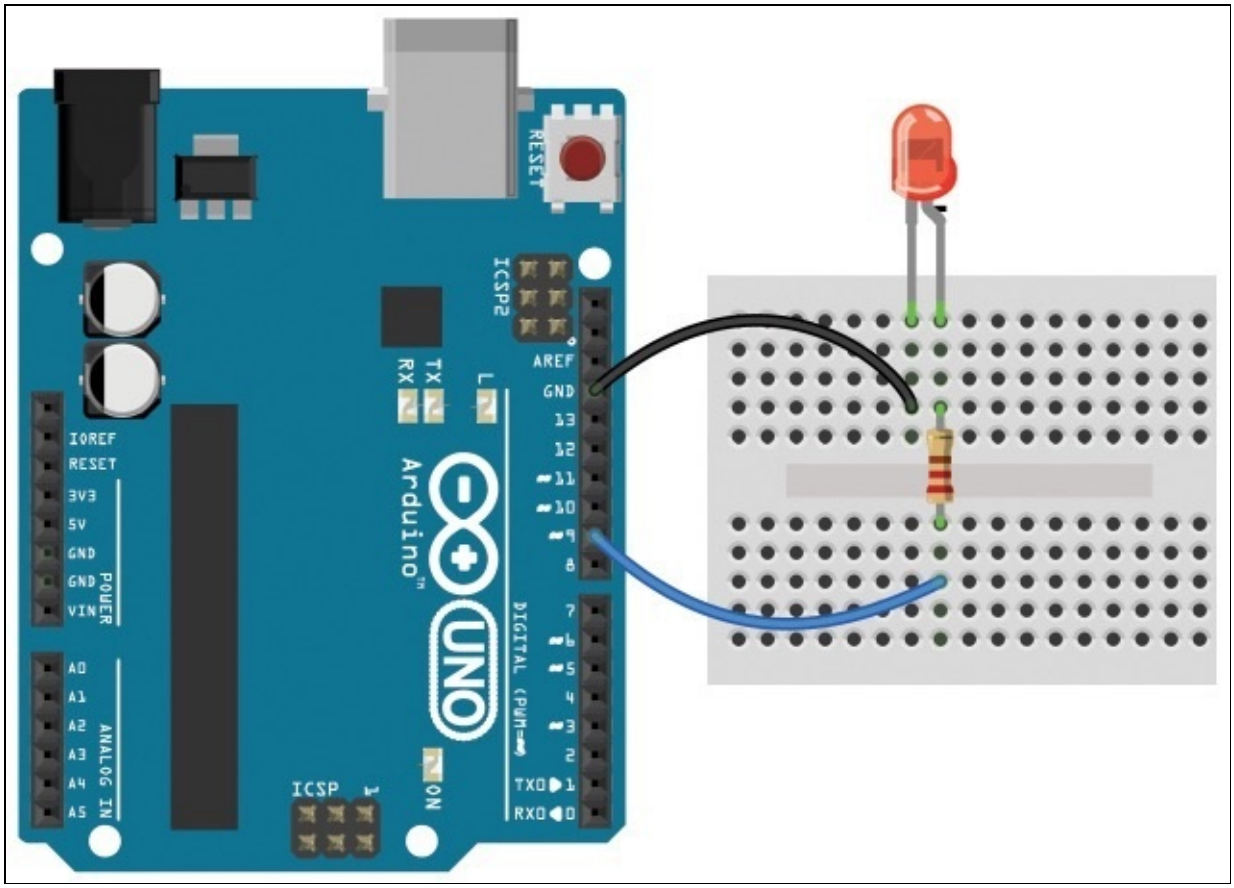


Figura 3.6 Il semplice schema di collegamento del LED al pin 9.

Segue la dichiarazione del pin di output. Come vedi è uno dei pin identificati dal simbolo `~` sulla scheda:

```
int ledPin = 9;    // LED connected to digital pin 9
```

La funzione `setup()`, pur non contenendo istruzioni, viene dichiarata per evitare errori durante la compilazione, e si passa subito alla funzione `loop()`:

```
void loop()  {
  // fade in from min to max in increments of 5 points:
  for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {
  // sets the value (range from 0 to 255):
  analogWrite(ledPin, fadeValue);
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
  }

  // fade out from max to min in increments of 5 points:
  for(int fadeValue = 255; fadeValue >= 0; fadeValue -=5) {
  // sets the value (range from 0 to 255):
  analogWrite(ledPin, fadeValue);
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
  }
}
```

Troviamo due blocchi di istruzioni molto simili racchiusi in due cicli `for()`. Analizziamo la dichiarazione del primo:

```
for(int fadeValue = 0; fadeValue <= 255; fadeValue +=5) {
```

Alla variabile `fadeValue` viene attribuito un valore iniziale di `0`, e il ciclo viene ripetuto fino a quando, con incrementi di `5`, la variabile non assume un valore maggiore di `255`.

> **NOTA** *Nell'*Appendice B *trovi la descrizione dettagliata delle strutture di controllo che puoi utilizzare nei tuoi sketch, tra cui il ciclo* `for`.

All'interno del ciclo l'istruzione `analogWrite()` agisce sulla luminosità del LED collegato al pin `9`, mentre `delay()` rallenta l'esecuzione, permettendoci di vedere l'effetto di *fading*.

Quando il valore della variabile `fadeValue` supera `255` la condizione non è più verificata e lo sketch passa oltre, incontrando il successivo ciclo `for`, analogo, che a decrementi di `5` riporta gradualmente la luminosità del LED a `0`.

Terminato anche il secondo ciclo il LED è tornato allo stato iniziale, e siamo giunti al termine della funzione `loop()`.

Verifica lo sketch ed esegui l'upload per osservare il comportamento del LED collegato sulla breadboard. Con i valori definiti nel codice il LED compie un ciclo completo in circa tre secondi. Modificando i valori numerici nello sketch possiamo agire sull'effetto.

Modifiche all'effetto fading

Applichiamo le tecniche imparate nei capitoli precedenti anche a questo sketch: con l'utilizzo delle costanti possiamo rendere più semplice la lettura del codice e più veloce la modifica dei parametri.

Rispetto allo sketch originale, nella sezione iniziale dichiariamo anche le costanti legate all'ampiezza degli incrementi nel ciclo e alla durata della pausa tra ogni iterazione:

```
/*
Sketch Fading modificato con l'introduzione delle costanti
*/

// dichiarazione delle costanti:
const int ledPin = 9;
// pin PWM a cui è collegato il LED
const int incremento = 5;
// ampiezza di ogni step
const int intervallo = 30;
// pausa in millisecondi tra le iterazioni

void setup()  {
  // dichiarazione di ledPin come output
  pinMode(ledPin, OUTPUT);
}

void loop()  {
  // aumento graduale della luminosità:
  for(int fadeValue = 0; fadeValue <= 255; fadeValue += incremento) {
    // trasmissione del valore PWM al pin
    analogWrite(ledPin, fadeValue);
    // pausa
    delay(intervallo);
  }

  // diminuzione graduale della luminosità:
  for(int fadeValue = 255; fadeValue >= 0; fadeValue -= incremento) {
    // trasmissione del valore PWM al pin
    analogWrite(ledPin, fadeValue);
    // pausa
    delay(intervallo);
  }
}
```

All'interno delle istruzioni nel `loop()` riprendiamo le costanti definite all'inizio, sostituendole ai valori hardcoded. Il resto delle istruzioni non viene modificato in alcun modo.

Agendo solo sulla sezione iniziale possiamo intervenire sui parametri che caratterizzano l'effetto. Modificando il valore di `incremento` decidiamo quanto varia la luminosità a ogni iterazione, mentre con `intervallo` impostiamo la pausa tra le ripetizioni.

Con `incremento = 5` e `intervallo = 30` l'effetto sarà identico allo sketch originale. Prova a modificare questi valori e carica lo sketch sulla scheda: osserva gli effetti prodotti sul LED collegato al pin `9`.

NOTA *Fai una prova impostando il valore* `incremento = 64` *e il valore* `intervallo = 500`*. Un salto così ampio tra i valori di luminosità e una pausa di mezzo secondo tra le iterazioni rendono*

percepibile a occhio nudo l'effetto prima mascherato dalla velocità: il LED raggiunge "gradini" di luminosità in sequenza. Con un `intervallo` *minore l'effetto complessivo è una sfumatura continua.*

Passiamo ora ad analizzare una delle potenzialità maggiori di Arduino: la lettura di dati in ingresso provenienti da sensori e componenti collegati alla scheda. Abbiamo già fatto qualche esperimento con un pulsante nei capitoli precedenti, ma le possibilità offerte dall'elaborazione di input analogici aprono scenari più complessi e interessanti.

Gli input analogici: il convertitore A/D

Parlando di dati in input acquisiti da sensori che leggono l'ambiente intorno alla scheda, spesso ci troviamo di fronte a informazioni per le quali il livello di precisione necessario è superiore al semplice `HIGH` e `LOW` trasmesso da un pulsante. Il microcontrollore della scheda Arduino Uno dispone di un convertitore A/D (analogico-digitale) a 6 canali collegato ai pin da `A0` ad `A5`. Questo sistema ha una risoluzione di 10 bit: è in grado di ricondurre un fenomeno elettrico a un valore digitale compreso tra `0` e `1023` in base al voltaggio letto.

L'istruzione `analogRead()` precede come unico parametro il pin da cui leggere l'informazione convertita relativa al voltaggio letto.

Analizziamo lo sketch **01.Basic** > **AnalogReadSerial**, un semplice sketch per leggere il valore trasmesso da un sensore, in questo caso un potenziometro, e comunicarlo tramite la connessione seriale.

Esistono potenziometri di varie forme e dimensioni, da quelli rotativi (le manopole del volume delle radio) a quelli lineari (le regolazioni dei canali nei mixer audio, per esempio), ma tutti condividono lo stesso principio di funzionamento: offrono una resistenza variabile al passaggio della corrente a seconda della posizione assunta dal cursore (**Figura 3.7**).

Come negli altri casi fin qui descritti, nella prima parte viene descritto il circuito da realizzare per eseguire il test:

```
Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.
```

Nell'esempio viene utilizzato un potenziometro rotativo che dispone di tre pin. Collegando i due pin esterni rispettivamente a `5V` e `GND` sulla scheda, la corrente trasferita al pin centrale, da collegare ad `A0` per la lettura, varierà in base alla rotazione.

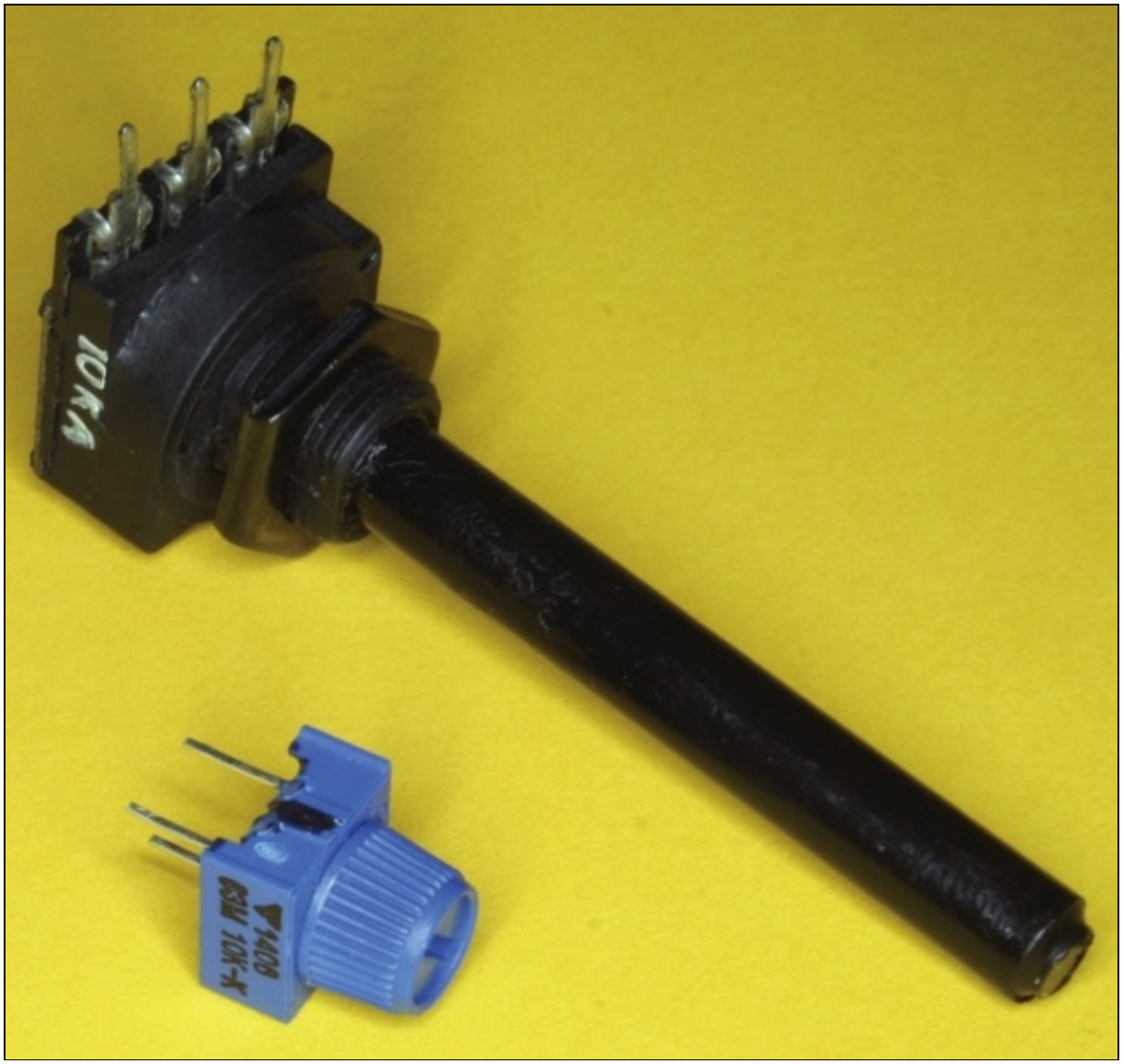


Figura 3.7 Un potenziometro rotativo.

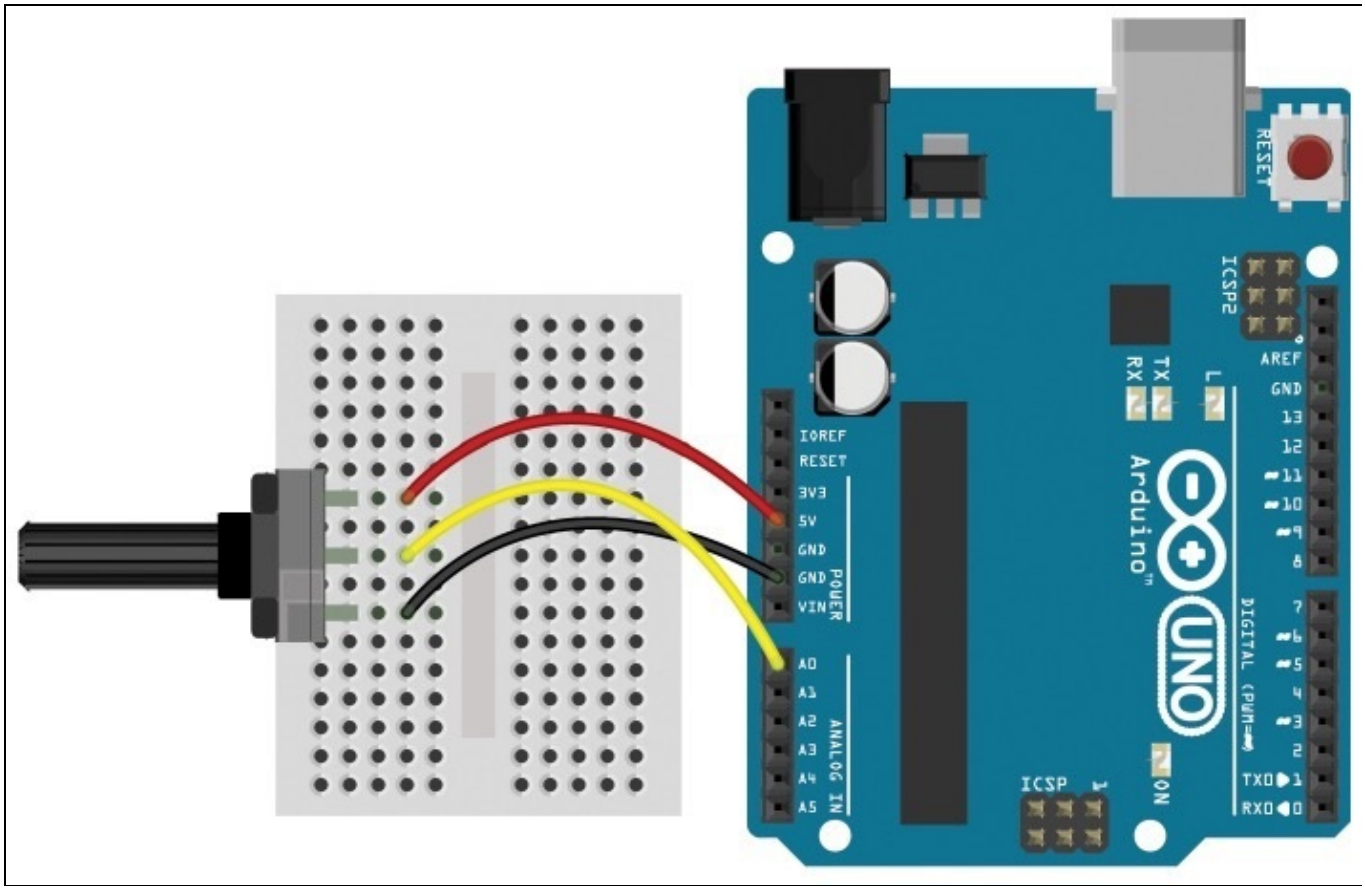


Figura 3.8 Il potenziometro collegato ad Arduino Uno: i due pin esterni sono collegati rispettivamente a 5V e GND, il pin centrale ad A0.

La funzione `setup()` si preoccupa di abilitare la connessione seriale:

```
void setup() {
  // initialize serial communication at 9600 bps:
  Serial.begin(9600);
}
```

NOTA *Non è necessario definire il pin* `A0` *come ingresso, poiché nel caso dei pin analogici è il comportamento di default.*

La funzione `loop()` raggruppa le istruzioni principali:

```
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);
  // delay in between reads for stability
}
```

Con `analogRead()` leggiamo il valore e lo attribuiamo alla variabile `sensorValue`. Con l'istruzione `Serial.println()` comunichiamo il valore tramite la connessione seriale. Un'ultima istruzione `delay(1)`, impercettibile, rende più stabile l'esecuzione dello sketch.

Verifichiamo la correttezza del codice e trasferiamo lo sketch sulla scheda. Aprendo il monitor seriale vedremo una lista di numeri in continuo aggiornamento: sono i voltaggi letti dal pin `A0` e convertiti in valori digitali dal microcontrollore della scheda. Prova a ruotare la manopola: i valori cresceranno o diminuiranno a seconda del senso di rotazione, per arrivare agli estremi `0` (0 Volt) e `1023` (5 Volt; **Figura 3.9**).

Verificato il funzionamento della manopola, proviamo a utilizzare il valore letto per agire sulla luminosità di un LED.

NOTA *Per un esempio di interazione differente tra input e output, dai un'occhiata allo sketch* 03.Analog > AnalogInput *tra gli esempi dell'IDE: in questo caso il valore letto dal pin analogico viene utilizzato per definire la frequenza con cui il LED collegato al pin* `13` *lampeggia: più basso è il valore, più brevi saranno gli intervalli.*

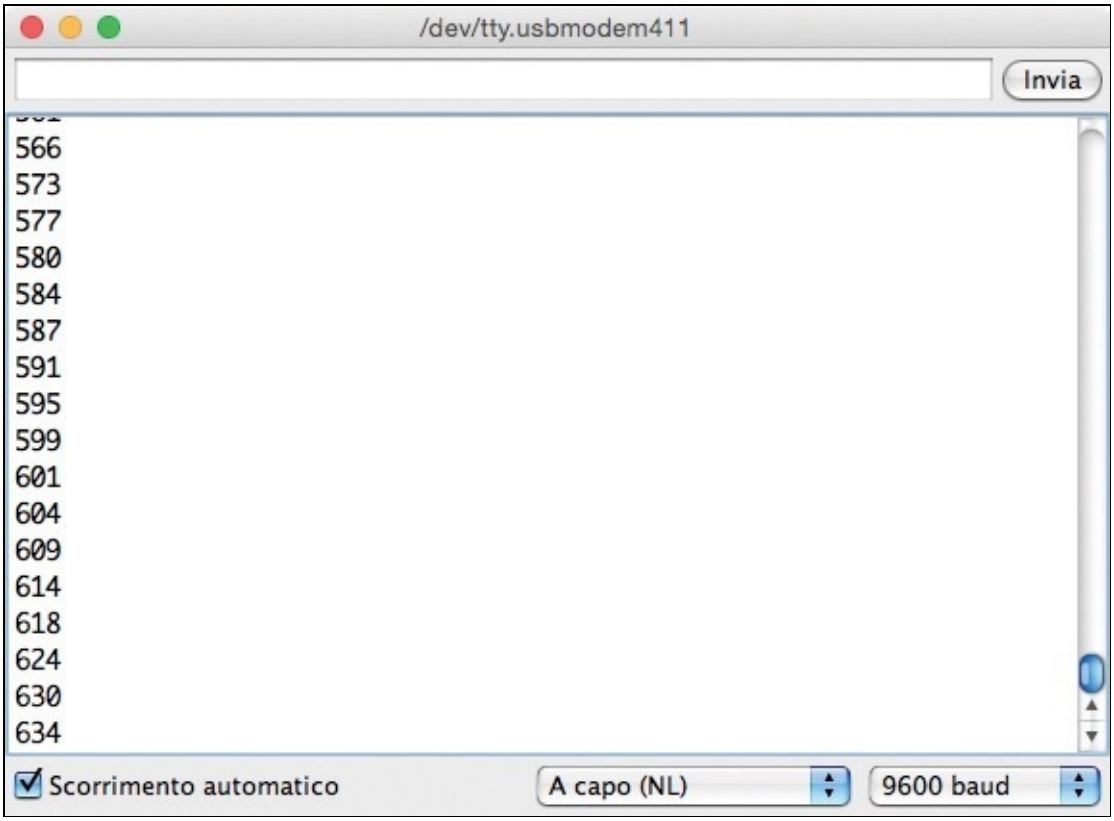


Figura 3.9 Ruotando la manopola del potenziometro il valore letto dal pin A0 varia di conseguenza.

I CONTROLLER ANALOGICI

Ti sei mai chiesto come funzionano i controller analogici presenti sui joystick di molte console? Esattamente secondo questo principio: la manopola è collegata a due potenziometri posti a 90° uno rispetto all'altro. Questa configurazione permette di poter calcolare in ogni momento con estrema semplicità la posizione precisa della manopola: è sufficiente combinare l'informazione dell'asse `X` con quella dell'asse `Y`. Nella figura puoi notare che ognuno dei due potenziometri ha tre pin, proprio come quello utilizzato negli esempi di queste pagine. La tecnica per leggere il dato sarà analoga a quella utilizzata nel nostro sketch di esempio.

Input e output analogici insieme

Proviamo a combinare quello che abbiamo imparato fin qui sul convertitore A/D in ingresso e sull'output PWM: realizziamo un circuito, e il relativo sketch, che in base al valore acquisito da un sensore imposta la luminosità di un LED.

Come abbiamo illustrato nelle pagine precedenti, provvediamo al collegamento con la scheda Arduino di un potenziometro, per leggere dati in ingresso con il pin `A0`, e di un LED su un pin dotato di funzionalità PWM, il pin digitale `9`.

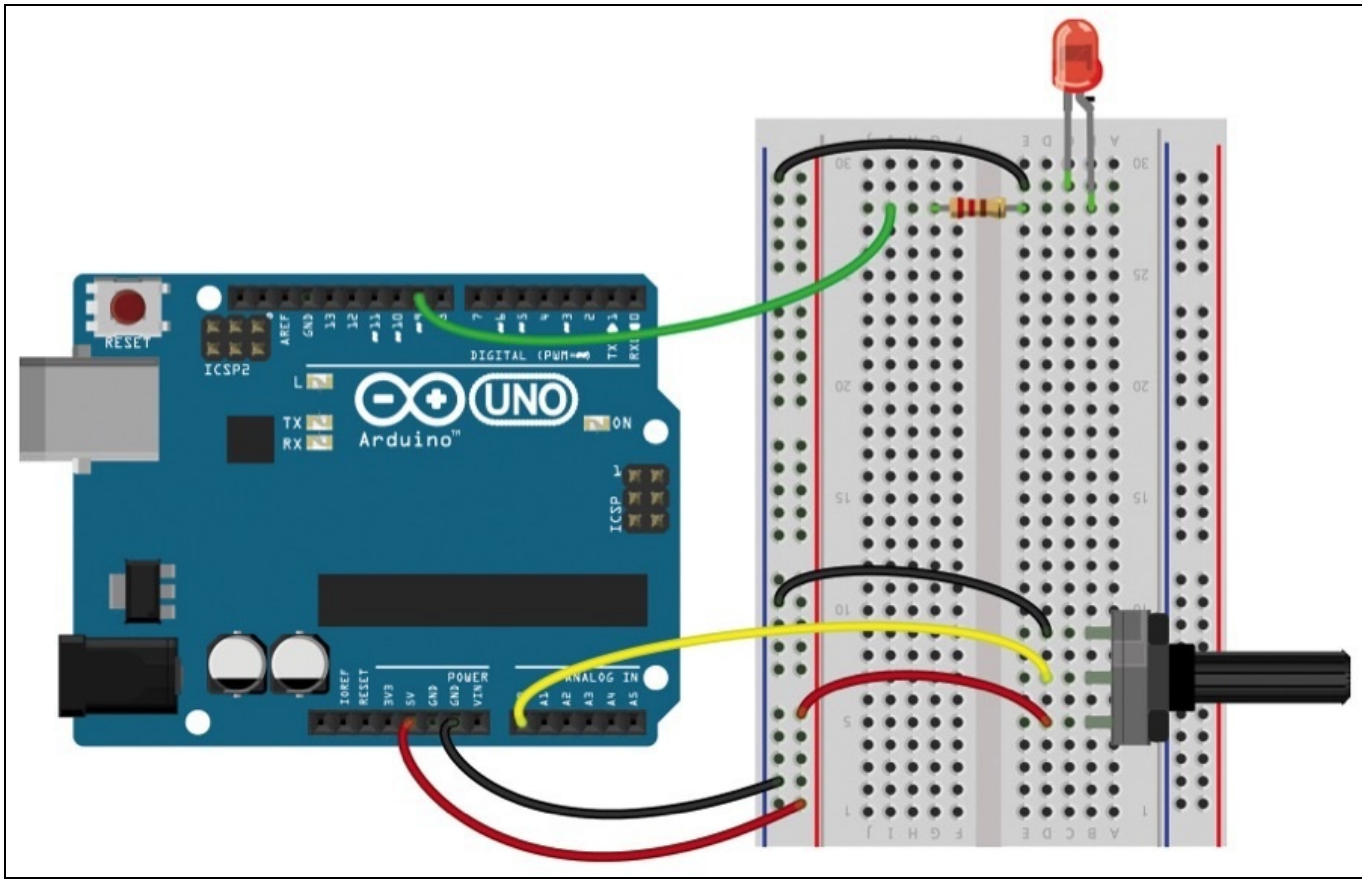


Figura 3.10 Connetti un LED al pin 9 e un potenziometro al pin A0. Per semplificare il circuito, 5V e GND sono collegati alle due fasce di pin sulla sinistra della breadboard.

Apportiamo alcune modifiche allo sketch descritto nelle pagine precedenti, in cui leggiamo il valore trasmesso dal potenziometro: ora non solo comunicheremo il dato tramite la porta seriale, ma lo utilizzeremo direttamente sulla scheda per agire sulla luminosità del LED:

```
/*
Input analogico e output PWM
*/

// dichiarazione delle costanti:
const int sensorPin = A0;
// il pin di input del sensore
const int ledPin = 9;
// il pin di output PWM per il LED
```

```
// dichiarazione delle variabili:
int sensorValue = 0;
// variabile per memorizzare i dati in ingresso
int ledValue = 0;
// variabile per il valore della luminosità

void setup() {
  // impostazione del pin in output
  pinMode(ledPin, OUTPUT);
  // inizializzazione della connessione seriale
  Serial.begin(9600);
}

void loop() {
  // lettura del valore del sensore
  sensorValue = analogRead(sensorPin);
  // conversione della lettura in valori da 0 a 255
  ledValue = map(sensorValue, 0, 1023, 0, 255);
  // impostazione del segnale PWM su pin del LED
  analogWrite(ledPin, ledValue);
  // comunicazione seriale della lettura e del valore PWM
  Serial.print(sensorValue);
  Serial.print("\t");
  Serial.println(ledValue);
  // delay per la stabilità del programma
  delay(1);
}
```

Nella prima parte dichiariamo le costanti e successivamente le variabili, inclusa `ledValue` che utilizzeremo nel `loop()` per memorizzare il valore di luminosità da attribuire al LED.

La funzione `setup()`, che prima si occupava di avviare la comunicazione seriale, ora imposta anche il comportamento del pin a cui è collegato il LED su `OUTPUT`.

In particolare nel `loop()` troviamo l'istruzione `map()`, molto utile quando si lavora con sensori analogici: la sua sintassi prevede cinque parametri:

- il valore da convertire, nel nostro caso `sensorValue`;
- il limite inferiore dei possibili valori in ingresso;
- il limite superiore dei possibili valori in ingresso;
- il limite inferiore dei valori in uscita;
- il limite superiore dei valori in uscita.

Questa funzione permette di semplificare la conversione dei dati da un intervallo di valori a un altro: nel caso specifico i pin analogici restituiscono valori compresi tra `0` e `1023`, mentre i pin PWM in output prevedono valori compresi tra `0` e `255`. Con questa singola istruzione eseguiamo una proporzione lineare tra l'intervallo di valori in ingresso e quello di uscita, convertendo il dato letto in uno adeguato per l'output PWM.

Da notare, nelle istruzioni finali che si occupano della comunicazione seriale, l'utilizzo di `\t`: nel monitor seriale verrà mostrato come una tabulazione, utile per

mantenere incolonnati i valori visualizzati.

Verifica lo sketch e trasferiscilo sulla scheda: ruotando la manopola il LED modificherà la propria luminosità, e nel monitor seriale vedrai visualizzati i valori letti dal pin `A0` e trasmessi al pin `9`.

Collegare una fotoresistenza

Proviamo ora a lavorare con un altro tipo di sensore: una *fotoresistenza* (in inglese abbreviata *LDR*). Come ben descritto dal nome, questo componente varia la propria resistenza al passaggio della corrente in base alla quantità di luce che lo colpisce. Vediamo come collegarlo alla scheda Arduino per utilizzarlo.

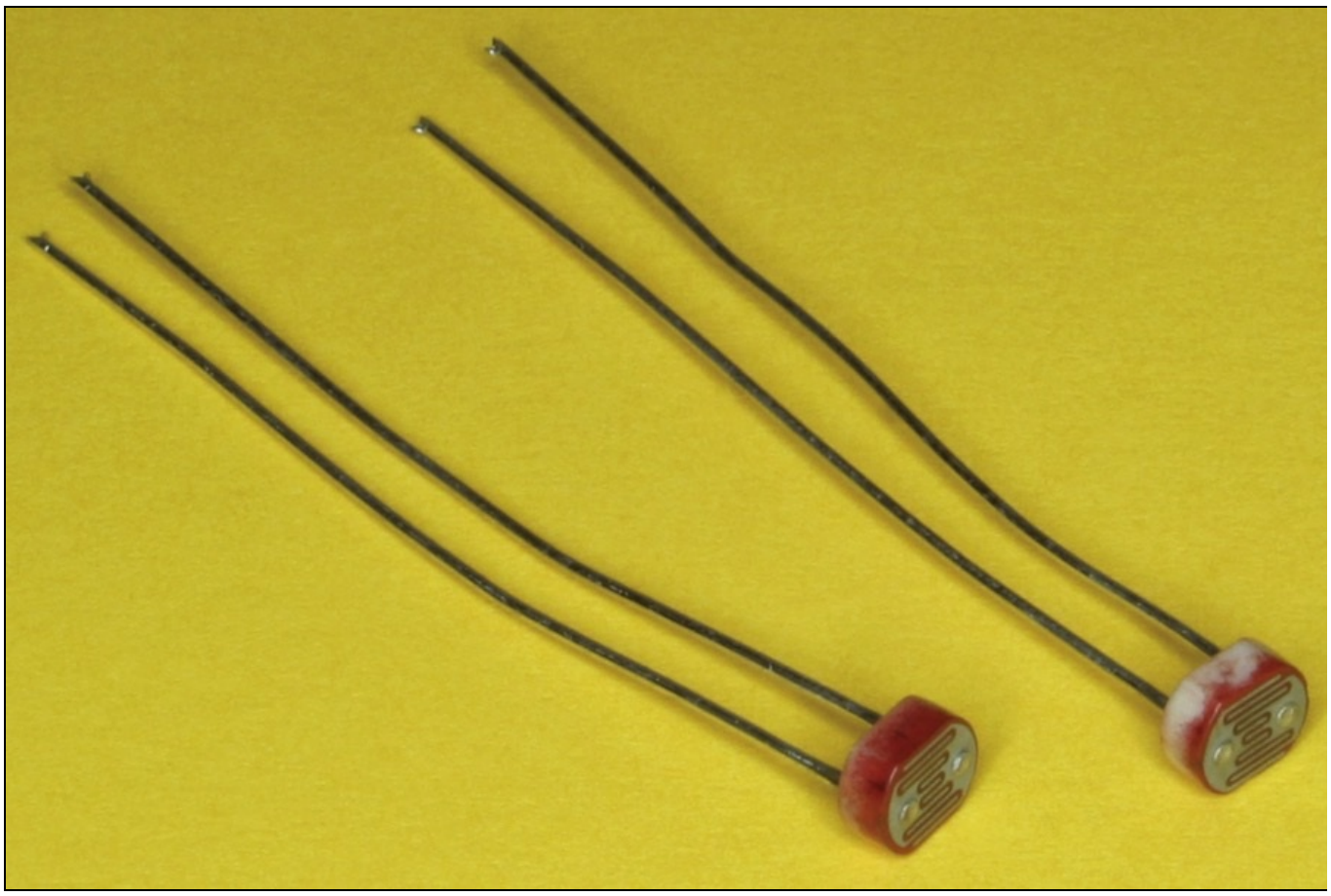


Figura 3.11 Due fotoresistenze; la parte superiore del componente è quella sensibile alla luce.

Per leggere il dato analogico rilevato dal sensore, in questo caso è necessario realizzare un circuito definito *partitore di tensione*.

Connetti uno dei terminali della fotoresistenza al pin `5V` e l'altro al pin `GND` ponendo in serie un resistore con un valore elevato, nell'ordine dei 10k Ohm. Tra il resistore e la fotoresistenza collega un jumper al pin `A0`. In questo modo il pin potrà leggere la caduta di tensione legata all'assorbimento variabile della fotoresistenza.

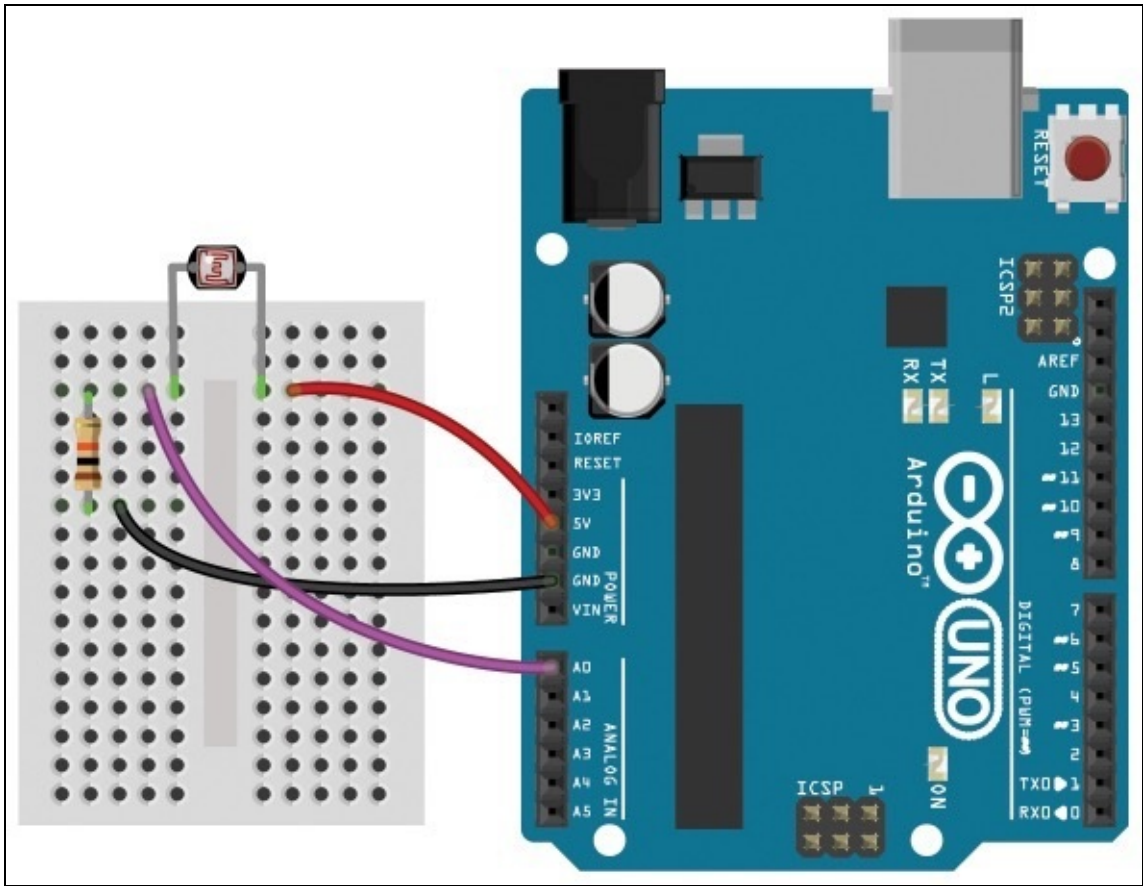


Figura 3.12 La fotoresistenza collegata al pin A0 della scheda.

Il circuito è completo; proviamo a caricare sulla scheda lo sketch di esempio **01.Basic** > **AnalogReadSerial**, già utilizzato in precedenza.

> **NOTA** *Lo sketch si occupa solo di leggere il voltaggio al pin* `A0`*: sostituire il potenziometro con una fotoresistenza non interferisce con l'esecuzione delle istruzioni.*

Aprendo il monitor seriale possiamo leggere i valori ricevuti dal pin `A0`. Prova a oscurare il sensore con una mano: il valore scenderà; posiziona una fonte luminosa vicino al sensore e la lettura sarà più alta.

Sperimentando con diversi livelli di luce possiamo subito notare alcune differenze rispetto all'esperimento con il potenziometro.

- I dati sono meno stabili, poiché la fotoresistenza è sensibile a cambiamenti di luce anche minimi.
- A seconda dell'ambiente in cui ci troviamo i valori massimi e minimi che potremo leggere saranno differenti.
- Per raggiungere i valori `0` e `1023` dovremo impegnarci con l'illuminazione o la schermatura.

Queste considerazioni portano alla necessità di una taratura del sensore: nello sketch è necessario prevedere degli accorgimenti per "ripulire" il segnale in ingresso

e renderlo più utilizzabile.

Ipotizziamo di voler utilizzare l'input, come nell'esempio con il potenziometro, per intervenire sulla luminosità di un LED. Iniziamo modificando lo sketch di comunicazione seriale, introducendo la funzione `map()` già illustrata per verificare prima di tutto il dato in output:

```
/*
Sketch per lettura e taratura di una fotoresistenza
*/

// dichiarazione delle costanti:
const int sensorPin = A0;      // il pin di input
                               // del sensore

// dichiarazione delle variabili:
int sensorValue = 0;           // variabile per memorizzare
                               // i dati in ingresso
int parsedValue = 0;           // variabile per memorizzare
                               // il dato ripulito

void setup() {
  // inizializzazione della comunicazione seriale
  Serial.begin(9600);
}

void loop() {
  // lettura del valore dal sensore
  sensorValue = analogRead(sensorPin);

  // mappatura del valore letto
  parsedValue = map(sensorValue, 140, 810, 0, 255);
  parsedValue = constrain(parsedValue, 0, 255);

  // comunicazione del valore tramite seriale
  Serial.println(parsedValue);

  // delay per la stabilità del programma
  delay(1);
}
```

Possiamo definire i limiti superiore e inferiore dell'intervallo di lettura in base ai dati trasmessi dal monitor seriale: nell'esempio `140` e `810` sono i valori minimo e massimo rilevati in condizioni normali. Ci prepariamo già da questo passaggio all'obiettivo finale dello sketch, ovvero inviare un segnale PWM al LED, mappando il valore letto in un intervallo compreso tra `0` e `255`. Con l'istruzione `constrain()` dopo la mappatura ci assicuriamo che eventuali dati anonimi, esterni all'intervallo che abbiamo individuato sperimentalmente, non generino valori fuori scala.

Verifica lo sketch e caricalo sulla scheda. Ora nel monitor seriale leggerai i valori mappati della luminosità. Interagendo con il sensore ti accorgerai che rispetto a prima sarà più semplice arrivare agli estremi `0` e `255` dell'intervallo.

Quando sei soddisfatto dei valori letti tramite il monitor seriale e della taratura puoi passare alla fase successiva: collegare un LED al pin `9` della scheda per agire sulla sua luminosità.

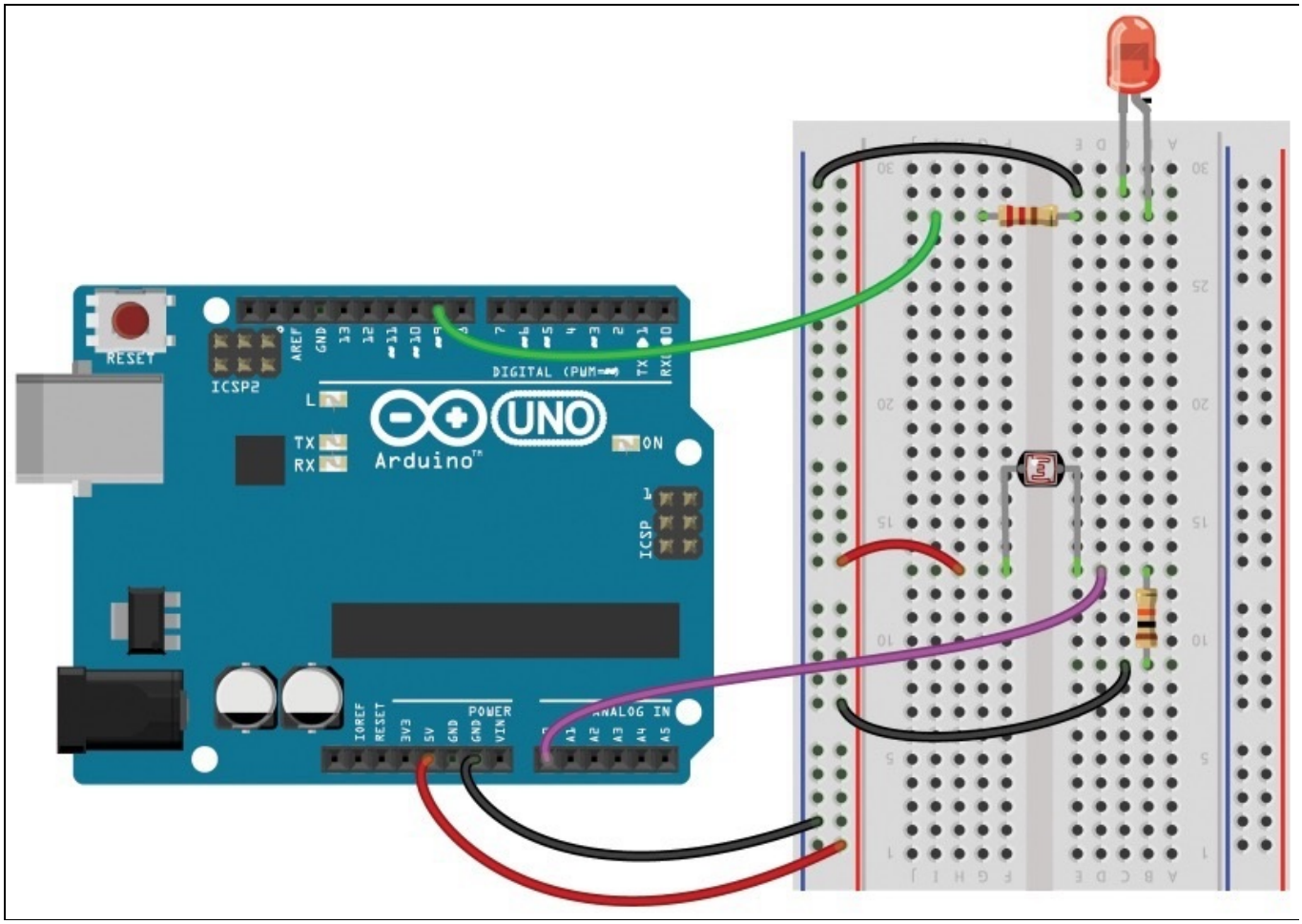


Figura 3.13 Riprendendo i circuiti della Figura 3.10, il LED è collegato al pin digitale 9 e la fotoresistenza al pin analogico A0.

Lo sketch riprende in tutto e per tutto quello utilizzato per l'interazione con il potenziometro: l'unico dettaglio è l'aggiustamento dei parametri dell'istruzione `map()` e l'introduzione di `constrain()`:

```
/*
  Input analogico con taratura e output PWM
*/

// dichiarazione delle costanti:
const int sensorPin = A0;
// il pin di input del sensore
const int ledPin = 9;
// il pin di output PWM per il LED

// dichiarazione delle variabili:
int sensorValue = 0;
// variabile per memorizzare i dati in ingresso
int ledValue = 0;
// variabile per il valore della luminosità

void setup() {
  // impostazione del pin in output
  pinMode(ledPin, OUTPUT);
  // inizializzazione della connessione seriale
  Serial.begin(9600);
}

void loop() {
  // lettura del valore del sensore
  sensorValue = analogRead(sensorPin);
  // conversione della lettura in valori da 0 a 255
```

```
  ledValue = map(sensorValue, 140, 810, 0, 255);
  ledValue = constrain(ledValue, 0, 255);
  // impostazione del segnale PWM su pin del LED
  analogWrite(ledPin, ledValue);
  // comunicazione seriale della lettura e del valore PWM
  Serial.print(sensorValue);
  Serial.print("\t");
  Serial.println(ledValue);
  // delay per la stabilità del programma
  delay(1);
}
```

Verifica e carica lo sketch: riducendo la luce che colpisce il sensore si ridurrà anche la luminosità del LED, fino al completo spegnimento. Aumentando la luce aumenterà anche la luminosità, fino alla totale accensione.

NOTA *Trovi un esempio che illustra un sistema alternativo di taratura, dove i valori non sono hardcoded nello sketch, alla voce* 03.Analog > Calibration*.*

È forse più interessante modificare il comportamento dello sketch per avere il LED acceso quando la luminosità ambientale è ridotta e un progressivo spegnimento al crescere della luminosità ambientale. Come puoi ottenere questo effetto? È sufficiente intervenire sulla mappatura dei valori invertendo gli ultimi due parametri dell'istruzione `map()`: `map(sensorValue, 140, 810, 255, 0)`. Alle letture più basse (corrispondenti a bassa luminosità ambientale) corrisponderanno valori più vicini a `255` e quindi una maggiore luminosità del LED. Viceversa, alle letture più alte (maggiore luminosità ambientale) il segnale PWM inviato al LED sarà più debole, tendente a `0`.

Conclusione

In questo capitolo ci siamo concentrati su input e output analogici. Abbiamo iniziato utilizzando le funzionalità PWM dei pin digitali contrassegnati con il simbolo `~` sulla scheda Arduino: dopo aver chiarito che la tecnica PWM è un'approssimazione digitale di un effetto analogico, abbiamo approfondito l'utilizzo dell'istruzione `analogWrite()`. Siamo poi passati alla lettura di valori analogici: abbiamo collegato un sensore ai pin analogici della scheda (da `A0` ad `A5`) e, con `analogRead()` e `map()`, abbiamo imparato a utilizzare i dati in ingresso per interagire con i pin PWM in output.

Abbiamo anche toccato con mano un concetto molto importante della prototipazione con Arduino: hardware e software lavorano in simbiosi, ma sono due livelli indipendenti uno dall'altro.

- Modificando un circuito possiamo collegare componenti diversi ad Arduino, utilizzando lo stesso sketch caricato sulla scheda per interagire con essi: abbiamo sostituito il potenziometro con una fotoresistenza, ma lo sketch che comunicava le letture al monitor seriale è rimasto lo stesso.
- Modificando solo lo sketch, senza intervenire sul circuito, possiamo definire comportamenti differenti: nell'interazione con la fotoresistenza una semplice modifica ai parametri dell'istruzione `map()` può alterare radicalmente il comportamento del LED collegato alla scheda.

Un progetto completo: Knock

In questo capitolo affronteremo un progetto più complesso degli esempi introduttivi visti finora: potremo renderci conto di alcuni aspetti della sperimentazione da tenere in considerazione per ottenere un prototipo usabile e interattivo.

Illustreremo un sistema per comandare una sorgente luminosa in grado di accendersi e spegnersi a ogni doppio colpo battuto sulla superficie su cui è appoggiata. Questo progetto, a prima vista semplice, ci permetterà di conoscere nuovi componenti, approfondire alcuni concetti e affrontare diversi problemi legati all’interazione con il mondo fisico.

L’elemento piezoelettrico

Per prima cosa collegheremo alla scheda Arduino un elemento piezoelettrico. Questo componente è in grado di reagire alle vibrazioni percepite, generando impulsi elettrici più o meno forti in base all’intensità della sollecitazione.

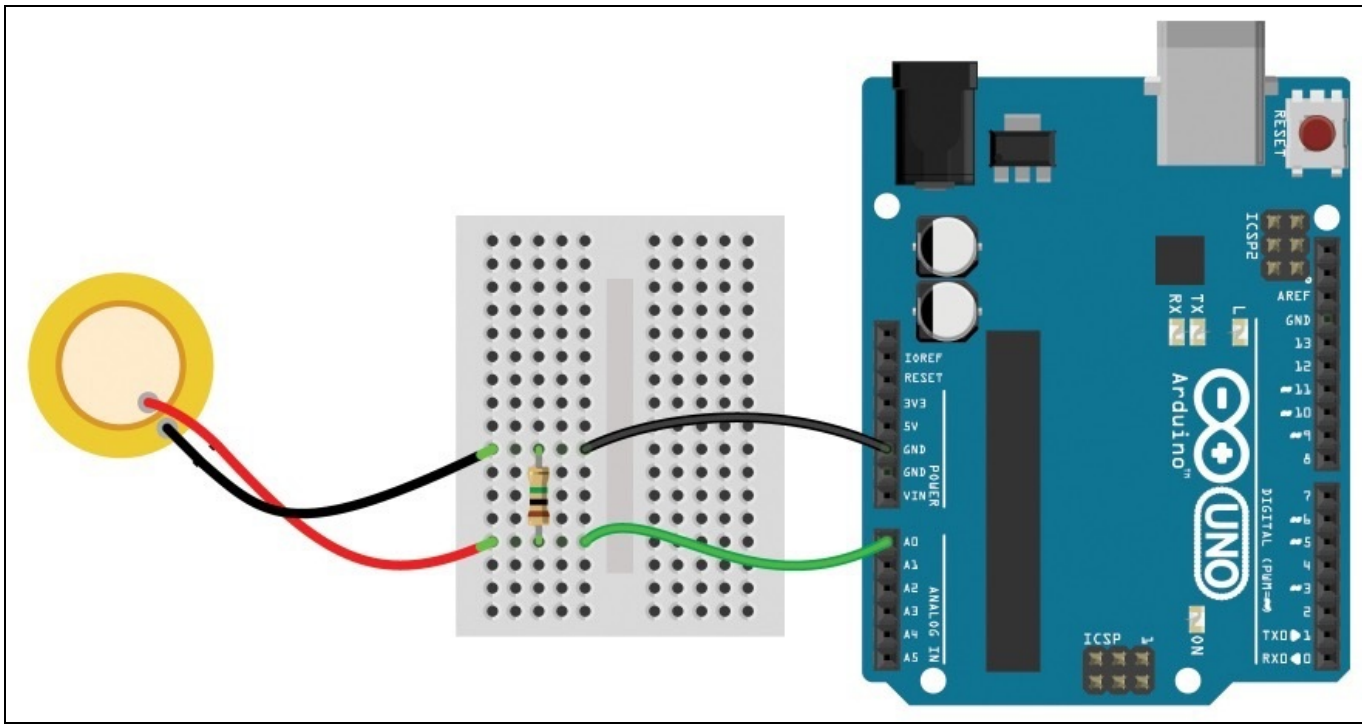


Figura 4.1 L’elemento piezoelettrico collegato al pin A0 della scheda.

Collega un polo del sensore al pin `GND` e l’altro al pin `A0`, mettendo in parallelo un resistore da 1 Mega Ohm, per proteggere la scheda da eventuali picchi o sovratensioni.

NOTA *Per consentire al sensore di percepire al meglio le vibrazioni della superficie fissalo al piano con del nastro adesivo o bloccalo con un peso; i risultati delle letture saranno più affidabili.*

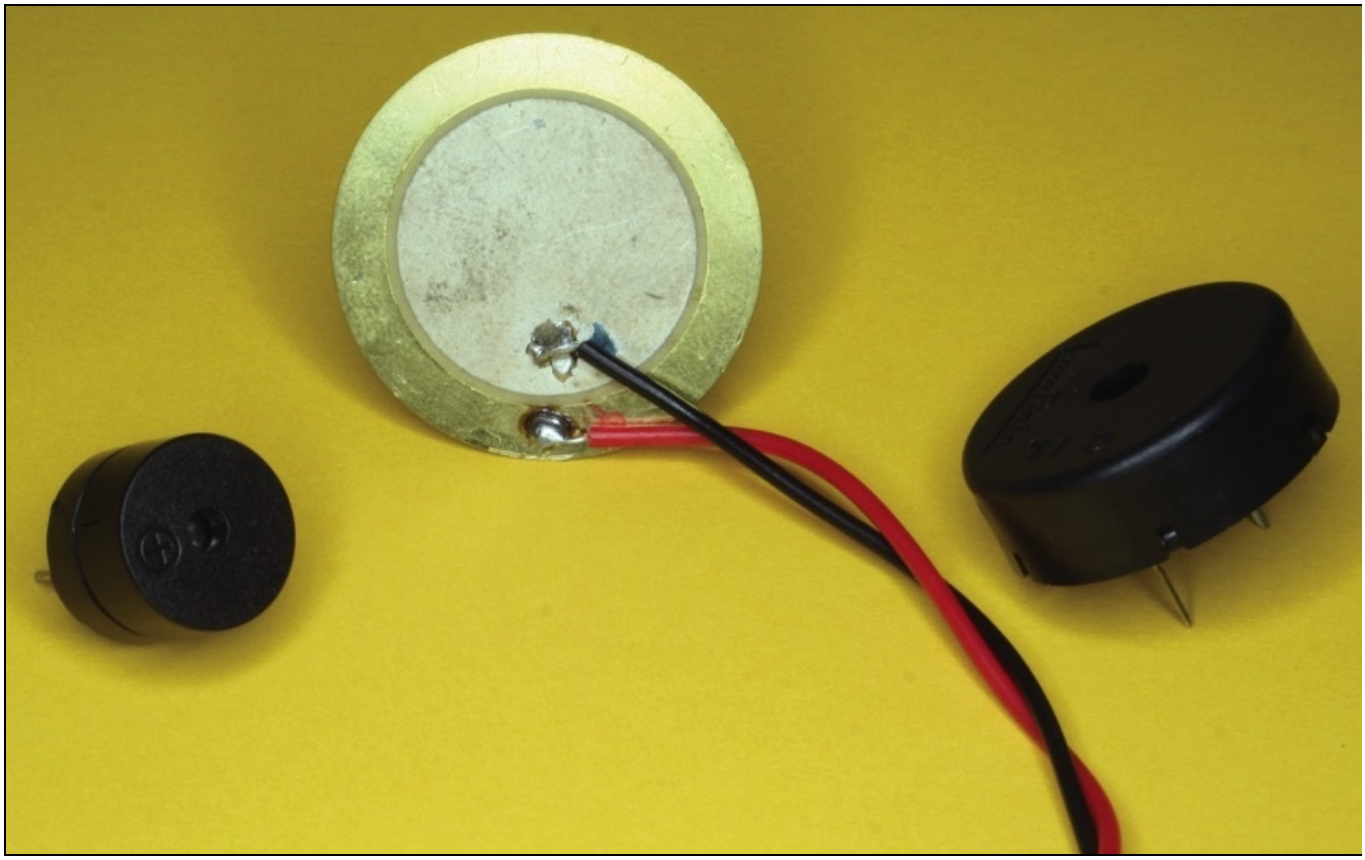


Figura 4.2 Alcuni sensori piezoelettrici: utilizzati come sensori trasformano le vibrazioni in segnali elettrici, come attuatori trasformano i segnali elettrici in vibrazioni (sonore).

Con lo sketch **01.Basic** > **AnalogReadSerial** già descritto nel capitolo precedente verifichiamo l'input ricevuto dal sensore attraverso il monitor seriale. Per comodità aggiungiamo però una condizione alla comunicazione del valore sulla porta seriale: saranno trasmessi solo i dati maggiori di `0`:

```
/*
Sketch per lettura di un pin analogico
*/

// dichiarazione delle costanti:
const int sensorPin = A0;
// il pin di input del sensore

// dichiarazione delle variabili:
int sensorValue = 0;
// variabile per memorizzare i dati in ingresso

void setup() {
  // inizializzazione della comunicazione seriale
  Serial.begin(9600);
}

void loop() {
  // lettura del valore dal sensore
  sensorValue = analogRead(sensorPin);
  // comunicazione del valore tramite seriale
  if (sensorValue > 0) Serial.println(sensorValue);
  // delay per la stabilità del programma
  delay(1);
}
```

NOTA *Quando associata alla condizione* `if()` *c'è una sola istruzione, come evidenziato in grassetto nello sketch, puoi omettere le parentesi graffe* `{` *e* `}` *che normalmente delimitano il ciclo.*

Dopo aver trasferito lo sketch sulla scheda Arduino e aver aperto il monitor seriale, prova a battere un colpo sul piano di lavoro su cui hai fissato il sensore: verrà visualizzata una lista di valori numerici. Fai qualche prova per familiarizzare con i valori registrati: a seconda dell'intensità delle vibrazioni questi saranno più o meno alti.

Per chiarezza cerchiamo di ripulire ulteriormente i dati registrati dal sensore, estraendo da ogni flusso di valori solo quelli significativi:

```
/*
Sketch per lettura di un sensore piezoelettrico
*/

// dichiarazione delle costanti:
const int sensorPin = A0;
// il pin di input del sensore

// dichiarazione delle variabili:
int sensorValue = 0;
// variabile per memorizzare i dati in ingresso
int maxInput = 0;
// variabile per memorizzare il dato più alto

void setup() {
  // inizializzazione della comunicazione seriale
  Serial.begin(9600);
}

void loop() {
  // lettura del valore dal sensore
  sensorValue = analogRead(sensorPin);
  // verifica del dato letto dal sensore
  if (maxInput < sensorValue) maxInput = sensorValue;
  // routine per riconoscere i singoli input
  if (sensorValue == 0 && maxInput > 0) {
    // comunicazione del valore tramite seriale
    Serial.println(maxInput);
    // reset della variabile maxInput
    maxInput = 0;
  }
  // delay per la stabilità del programma
  delay(1);
}
```

Inizializziamo una variabile `maxInput` che utilizzeremo nella funzione `loop()` per memorizzare il valore più alto di ogni sequenza di dati. Al termine di una sollecitazione c'è un valore significativo: verificata la condizione `if (sensorValue == 0 && maxInput > 0)`, comunichiamo la lettura al monitor seriale.

Trasferiamo lo sketch sulla scheda. Grazie agli accorgimenti presi verrà trasmesso al monitor seriale solo il valore massimo di ogni flusso di dati.

Notiamo però che a ogni colpo corrisponde più di un valore: i valori rilevati dal sensore creano false letture, soprattutto nella fase più debole dell'impulso. Proviamo a rifinire ulteriormente le letture prendendo spunto dalle routine di debounce dei dati letti alle quali abbiamo accennato nel **Capitolo 2**: se l'intervallo tra due sequenze di

dati è inferiore ad alcuni millisecondi, potremo considerare gli input appartenenti alla stessa sollecitazione:

```
/*
Sketch per lettura di un sensore piezoelettrico con debounce
*/

// dichiarazione delle costanti:
const int sensorPin = A0;
// il pin di input del sensore
const int ritardoLettura = 50;
// l'intervallo necessario per la pulizia del dato

// dichiarazione delle variabili:
int sensorValue = 0;
// variabile per memorizzare i dati in ingresso
int maxInput = 0;
// variabile per memorizzare il dato più alto

long tempoLettura = 0;
long intervalloLettura = 0;
// due variabili per la routine di pulizia del dato

void setup() {
  // inizializzazione della comunicazione seriale
  Serial.begin(9600);
}

void loop() {
  // lettura del valore dal sensore
  sensorValue = analogRead(sensorPin);
  // aggiornamento del riferimento temporale
  if (sensorValue > 0) tempoLettura = millis();
  // verifica del dato letto dal sensore
  if (maxInput < sensorValue) maxInput = sensorValue;
  // calcolo dell'intervallo di tempo
  intervalloLettura = millis() - tempoLettura;
  // routine eseguita solo quando l'input è terminato
  if (sensorValue == 0 && maxInput > 0 && intervalloLettura > ritardoLettura) {
    // comunicazione del valore tramite seriale
    Serial.println(maxInput);
    // reset della variabile maxInput
    maxInput = 0;
  }
  // delay per la stabilità del programma
  delay(1);
}
```

Abbiamo aggiunto una costante `ritardoLettura`, per definire l’intervallo di tempo all’interno del quale i dati vengono considerati come derivati alla stessa sollecitazione, e due variabili `tempoLettura` e `intervalloLettura`, che utilizzeremo nelle routine di verifica.

> **NOTA** *In base alla sensibilità del sensore che utilizzi potresti dover modificare il valore di* `ritardoLettura` *e adattarlo all’hardware che stai utilizzando.*

Con `if (sensorValue > 0) tempoLettura = millis()` memorizziamo a ogni input significativo un riferimento temporale per accorpare le letture. A ogni ciclo la variabile `intervalloLettura` assume come valore il numero di millisecondi trascorsi dall’ultima lettura utile del sensore, e questo dato viene utilizzato per integrare la condizione `if()` seguente: l’input è concluso e si procede alla comunicazione del

valore solo quando è trascorso un intervallo di tempo adeguato senza nuovi dati in ingresso.

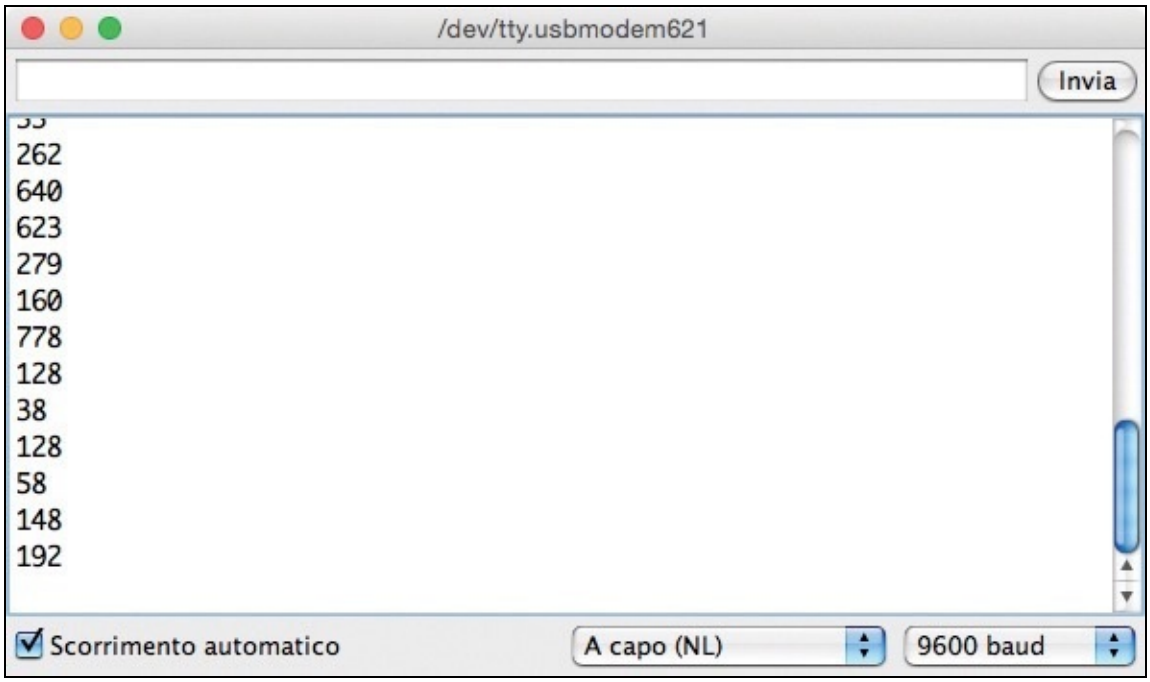


Figura 4.3 Il monitor seriale mostra l'intensità dei valori registrati. Puoi facilmente farti un'idea del tipo di dati con cui ha a che fare lo sketch.

Il concetto di soglia

A seconda della sensibilità del sensore che stai utilizzando ti sarai reso conto che vengono visualizzati anche dati corrispondenti a vibrazioni davvero leggere del piano sul quale è appoggiato. Per i nostri scopi abbiamo bisogno di distinguere i colpi intenzionali sul piano di lavoro dal “rumore di fondo” che deve essere ignorato. Optiamo per una soluzione versatile, utilizzando un potenziometro per la regolazione: in ogni momento potremo intervenire sulla taratura. Aggiungiamo il potenziometro sulla breadboard e colleghiamolo al pin A1 come abbiamo già visto negli esempi delle pagine precedenti.

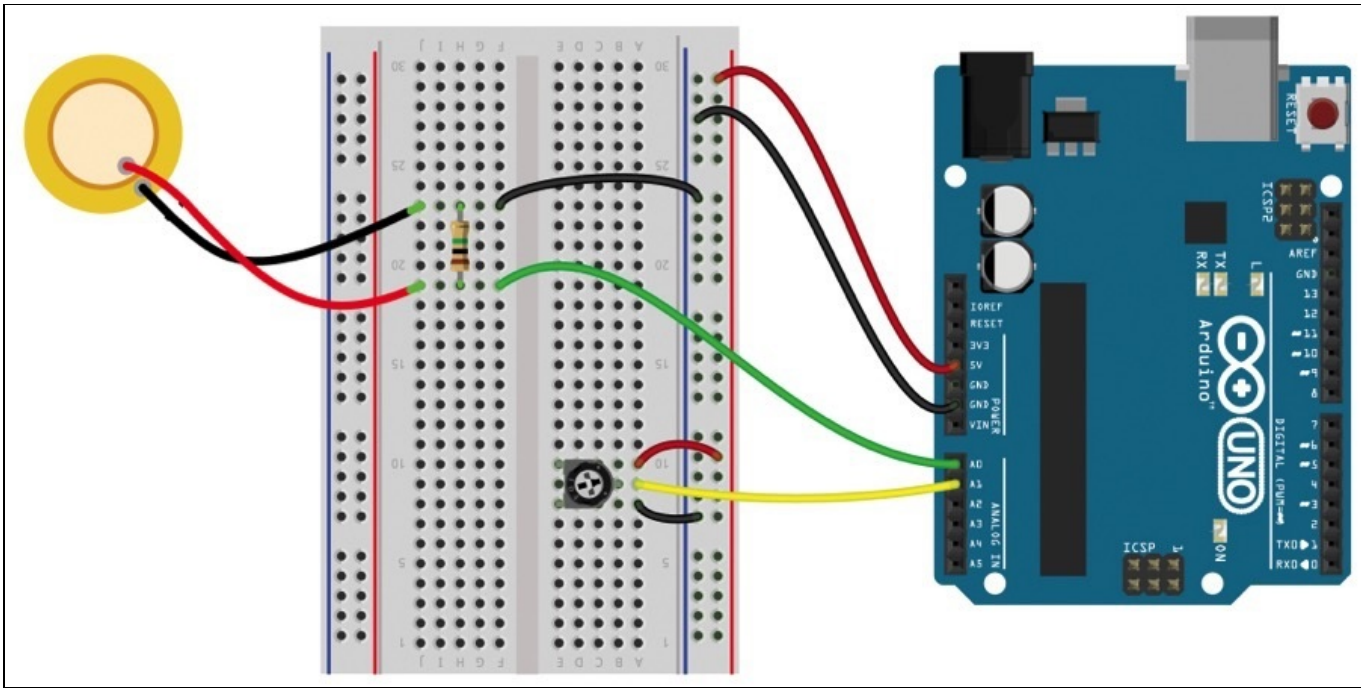
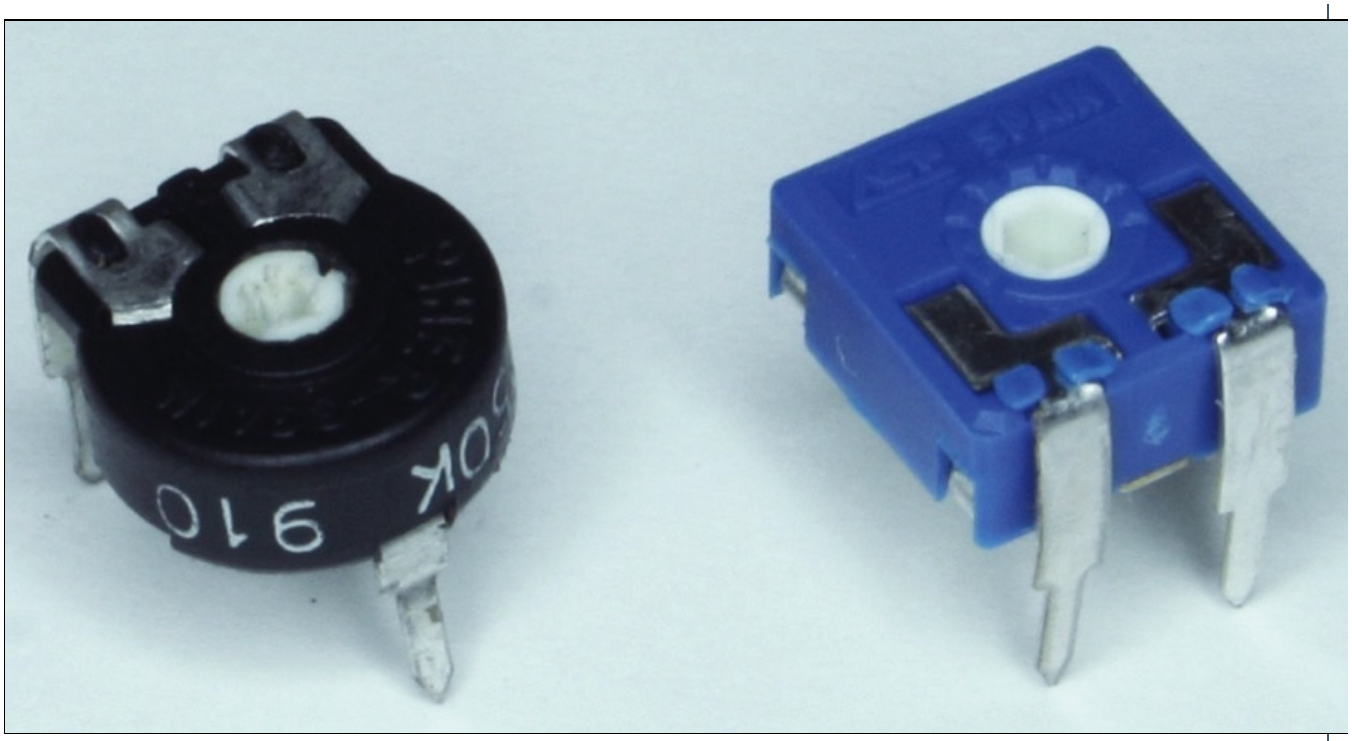


Figura 4.4 Il sensore piezoelettrico collegato al pin A0 e il potenziometro collegato al pin A1.

FORMA DIVERSA, STESSO COMPORTAMENTO

Il potenziometro mostrato nello schema della **Figura 4.4** è differente da quello utilizzato nel capitolo precedente solo nella forma: a volte chiamato *trimmer*, è più piccolo e richiede un cacciavite per la regolazione. Spesso questo tipo di potenziometro è preferibile quando la regolazione sarà poco frequente: il rischio di movimenti accidentali è infatti ridotto.



Modifichiamo lo sketch per tenere conto del valore letto dal potenziometro; in base a questa informazione comunicheremo o meno i dati sulla porta seriale:

```
/*
Sketch per lettura di un sensore piezoelettrico
con debounce
*/

// dichiarazione delle costanti:
const int sensorPin = A0;
// il pin di input del sensore
const int potPin = A1;
// il pin di input del potenziometro
const int ritardoLettura = 50;
// l'intervallo necessario per la pulizia del dato

// dichiarazione delle variabili:
int sensorValue = 0;
// variabile per memorizzare i dati in ingresso
int potValue;
// variabile per memorizzare la lettura del potenziometro
int maxInput = 0;
// variabile per memorizzare il dato più alto

long tempoLettura = 0;
long intervalloLettura = 0;
// due variabili per la routine di pulizia del dato

void setup() {
  // inizializzazione della comunicazione seriale
  Serial.begin(9600);
}

void loop() {
  // lettura del valore dai sensori
  sensorValue = analogRead(sensorPin);
  potValue = analogRead(potPin);
  // aggiornamento del riferimento temporale
  if (sensorValue > 0) tempoLettura = millis();
  // verifica del dato letto dal sensore
  if (maxInput < sensorValue) maxInput = sensorValue;
  // calcolo dell'intervallo di tempo
  intervalloLettura = millis() - tempoLettura;
  // routine eseguita solo quando l'input è terminato
  if (sensorValue == 0 && maxInput > 0 && intervalloLettura > ritardoLettura) {
```

```
    // comunicazione del valore superiore alla soglia
    if (maxInput > potValue) Serial.print(maxInput);
    // reset della variabile maxInput
    maxInput = 0;
  }
  // delay per la stabilità del programma
  delay(1);
}
```

Definite le variabili necessarie e le istruzioni per leggere il valore del potenziometro non resta che aggiungere una condizione con la quale verificare se il dato è da comunicare al monitor seriale: `if (maxInput > potValue) Serial.print(maxInput)`.

> **NOTA** *Non interveniamo sulla condizione combinata precedente per fare in modo che a ogni lettura il valore di* `maxInput` *venga riportato a* `0` *anche quando la lettura è inferiore al valore di soglia.*

Una volta caricato lo sketch sulla scheda prova a modificare la taratura del potenziometro: in base alla posizione assunta ti accorgerai che saranno necessarie sollecitazioni più o meno forti per attivare la comunicazione seriale.

LA REGOLAZIONE DEL POTENZIOMETRO

Sei curioso di conoscere la soglia a cui è regolato il potenziometro? Aggiungi l'istruzione `Serial.print(potValue)` all'interno dello sketch. Una buona posizione potrebbe essere nella condizione `if()` dove già viene comunicata la lettura:

```
if (maxInput > potValue) {
  Serial.print(maxInput);
  Serial.print(" > ");
  Serial.println(potValue);
}
```

Ogni lettura valida sarà accompagnata nel monitor seriale dal simbolo `>` (maggiore di) e dal valore di soglia impostato.

Due tocchi ravvicinati

Facciamo un altro passo avanti e raffiniamo ulteriormente lo sketch: reagiremo solo a due tocchi ravvicinati impostando un intervallo di tempo limite tra due letture consecutive. Abbiamo già incontrato la funzione `millis()`, che utilizzeremo anche in questo caso per valutare il tempo trascorso tra due input:

```
/*
Sketch per lettura di un sensore piezoelettrico - due colpi
*/

// dichiarazione delle costanti:
const int sensorPin = A0;
// il pin di input del sensore
const int potPin = A1;
// il pin di input del potenziometro
const int ritardoLettura = 50;
// l'intervallo necessario per la pulizia del dato
const int ritardoKnock = 500;
// l'intervallo per i due tocchi consecutivi

// dichiarazione delle variabili:
int sensorValue = 0;
// variabile per memorizzare i dati in ingresso
int potValue;
// variabile per memorizzare la lettura del potenziometro
int maxInput = 0;
// variabile per memorizzare il dato più alto

long tempoLettura = 0;
long intervalloLettura = 0;
// due variabili per la routine di pulizia del dato

long tempoKnock = 0;
long intervalloKnock = 0;
// due variabili per rilevare il doppio tocco

void setup() {
  // inizializzazione della comunicazione seriale
  Serial.begin(9600);
}

void loop() {
  // lettura del valore dai sensori
  sensorValue = analogRead(sensorPin);
  potValue = analogRead(potPin);
  // aggiornamento del riferimento temporale
  if (sensorValue > 0) tempoLettura = millis();
  // verifica del dato letto dal sensore
  if (maxInput < sensorValue) maxInput = sensorValue;
  // calcolo dell'intervallo di tempo
  intervalloLettura = millis() - tempoLettura;
  // routine eseguita solo quando l'input è terminato
  if (sensorValue == 0 && maxInput > 0 && intervalloLettura > ritardoLettura) {
    // comunicazione del valore superiore alla soglia
    if (maxInput > potValue) {
      intervalloKnock = millis() - tempoKnock;
      if (intervalloKnock < ritardoKnock) {
        Serial.println ("Knock knock!");
      }
      else {
        tempoKnock = millis();
      }
    }
    // reset della variabile maxInput
    maxInput = 0;
  }
  // delay per la stabilità del programma
```

```
  delay(1);
}
```

Definiamo la costante `ritardoKnock`, per impostare il valore in millisecondi al di sotto del quale considerare gli input consecutivi, e le variabili `tempoKnock` e `intervalloKnock`. Per misurare il tempo tra due tocchi (abbiamo già affinato l'algoritmo per isolare i singoli tocchi nei precedenti esempi) interveniamo all'interno della condizione `if (maxInput > potValue)`. Se il tempo trascorso è inferiore a quello definito con `ritardoKnock` comunichiamo al monitor seriale il messaggio **Knock knock!**, altrimenti la misurazione dell'intervallo riparte da zero.

Compila e trasferisci lo sketch sulla scheda. Apri il monitor seriale per visualizzare i messaggi inviati dalla scheda. Ricorda: hai sempre a disposizione il potenziometro per regolare la sensibilità del componente piezoelettrico, e puoi decidere la tolleranza del sistema modificando il valore della costante `ritardoKnock`.

> **NOTA** *Prova a inserire altre istruzioni* `Serial.print()` *per ricevere sul monitor seriale più informazioni sui valori assunti dalle variabili utilizzate: saranno dati preziosi per affinare il comportamento del prototipo.*

Dal monitor seriale a un output visibile

Una volta soddisfatti delle regolazioni modifichiamo il comportamento dello sketch: invece di inviare un messaggio tramite la porta seriale possiamo per esempio intervenire sul LED collegato al pin digitale `13`: al riconoscimento di un doppio tocco passerà da spento (`LOW`) ad acceso (`HIGH`) o viceversa.

Le modifiche da effettuare allo sketch non sono molte: come negli esempi dei capitoli precedenti, nella parte iniziale definiamo la costante `ledPin` per riferirci al pin a cui collegheremo il LED (nel nostro caso il pin `13`, con il LED già integrato sulla scheda):

```
const int ledPin = 13;
```

… e la variabile `ledValue` per memorizzare lo stato del pin:

```
int ledValue = LOW;
```

Non resta che definire il pin come output nella funzione `setup()`:

```
pinmode(ledPin, OUTPUT);
```

A questo punto nella funzione `loop()` dello sketch affianchiamo all'istruzione legata alla comunicazione seriale quelle necessarie per intervenire sull'output del pin:

```
if (maxInput > potValue) {
      intervalloKnock = millis() - tempoKnock;
      if (intervalloKnock < ritardoKnock) {
        ledValue = !ledValue;
        digitalWrite(ledPin, ledValue);
        Serial.println ("Knock knock!");
      }
      else {
        tempoKnock = millis();
      }
    }
```

Ogni volta che le condizioni saranno soddisfatte il valore di `ledValue` passerà da `LOW` a `HIGH` o viceversa, e lo stato di `ledPin` verrà aggiornato di conseguenza.

INPUT E OUTPUT

Abbiamo appena visto che è piuttosto semplice associare allo stesso input due comportamenti in output differenti: da un messaggio sul monitor seriale siamo passati a un output luminoso senza troppi stravolgimenti. Spesso nei tuoi sketch a uno stimolo corrisponderà una reazione, ma i due fenomeni (e le routine per acquisire dati e generare output) saranno, a livello di programmazione, indipendenti tra loro. Tieni sempre a mente questa considerazione, e cerca di scrivere codice ordinato e riutilizzabile.

Per completezza ecco lo sketch finale. Dopo averlo trasferito, con le istruzioni legate alla comunicazione seriale commentate, puoi anche scollegare la scheda dal

computer e alimentarla con una sorgente esterna; il funzionamento sarà identico:

```
/*
Sketch per lettura di un sensore piezoelettrico - LED
*/

// dichiarazione delle costanti:
const int sensorPin = A0;
// il pin di input del sensore
const int potPin = A1;
// il pin di input del potenziometro
const int ledPin = 13;
// il pin di output

const int ritardoLettura = 50;
// l'intervallo necessario per la pulizia del dato
const int ritardoKnock = 500;
// l'intervallo per i due tocchi consecutivi

// dichiarazione delle variabili:
int sensorValue = 0;
// variabile per memorizzare i dati in ingresso
int potValue;
// variabile per memorizzare la lettura del potenziometro
int maxInput = 0;
// variabile per memorizzare il dato più alto
int ledStato = LOW;
// variabile per memorizzare lo stato del pin

long tempoLettura = 0;
long intervalloLettura = 0;
// due variabili per la routine di pulizia del dato

long tempoKnock = 0;
long intervalloKnock = 0;
// due variabili per rilevare il doppio tocco

void setup() {
  pinMode(ledPin, OUTPUT);
  // inizializzazione della comunicazione seriale
  // Serial.begin(9600);
}

void loop() {
  // lettura del valore dai sensori
  sensorValue = analogRead(sensorPin);
  potValue = analogRead(potPin);
  // aggiornamento del riferimento temporale
  if (sensorValue > 0) tempoLettura = millis();
  // verifica del dato letto dal sensore
  if (maxInput < sensorValue) maxInput = sensorValue;
  // calcolo dell'intervallo di tempo
  intervalloLettura = millis() - tempoLettura;
  // routine eseguita solo quando l'input è terminato
  if (sensorValue == 0 && maxInput > 0 && intervalloLettura > ritardoLettura) {
    // comunicazione del valore superiore alla soglia
    if (maxInput > potValue) {
      intervalloKnock = millis() - tempoKnock;
      if (intervalloKnock < ritardoKnock) {
        ledStato = !ledStato;
        digitalWrite(ledPin, ledStato);
        Serial.println ("Knock knock!");
      }
      else {
        tempoKnock = millis();
      }
    }
    // reset della variabile maxInput
    maxInput = 0;
  }
  // delay per la stabilità del programma
  delay(1);
}
```

Comandare un carico elettrico più elevato

Lo sketch e il circuito che abbiamo realizzato funzionano come previsto: con un doppio colpo il LED si accende e con un altro si spegne. A questo punto perché non comandare una sorgente luminosa più generosa? Potrebbe essere lo spunto per realizzare una lampada da scrivania comandando dei LED ad alta potenza o addirittura una lampadina a 220V.

LA CORRENTE A 220V

Quando hai a che fare con correnti elevate presta moltissima attenzione: cavi scoperti o contatti errati possono provocare danni anche gravi a persone e cose. Puoi testare il comportamento di sketch e circuiti utilizzando componenti a bassa tensione. Se non sei più che sicuro di ciò che stai facendo, evita di eseguire operazioni anche solo potenzialmente rischiose, e chiedi aiuto a un esperto.

Il microcontrollore sulla scheda Arduino non può comandare direttamente correnti elevate; dovremo perciò ricorrere a un relè per evitare di sovraccaricare la scheda.

Come funziona un relè

Puoi pensare a un relè come a un interruttore acceso/spento, comandato da un impulso elettrico invece che dalla pressione di un pulsante: quando una corrente a bassa tensione passa nel circuito di comando, meccanicamente scatta una bobina che chiude il circuito principale, al quale è possibile collegare utilizzatori che assorbono molta potenza.

Togliendo tensione la bobina ritorna nella posizione iniziale, interrompendo il circuito.

NOTA *Dal datasheet puoi risalire alla corrente minima necessaria per attivare la bobina e a tutte le altre caratteristiche elettriche specifiche per il modello di relè che stai utilizzando.*

Con i relè possiamo comandare utilizzatori anche potenti (fino al limite determinato dalle caratteristiche del componente in uso) mantenendo la scheda Arduino elettricamente separata: non ci sarà rischio di sovraccarichi o sovratensioni che la potrebbero danneggiare.

Nel nostro caso il relè servirà per comandare una fonte luminosa di potenza maggiore del LED collegato al pin 13, una fonte non gestibile direttamente dalla scheda a causa dell’elevato assorbimento. Per evitare rischi, durante la fase di prototipazione continueremo a lavorare con tensioni e correnti ridotte per poi eventualmente passare a carichi più elevati alla fine dei test: il consiglio è quello di

collegare al relè un’alimentazione elettrica separata dalla scheda Arduino (per esempio una batteria) e un semplice LED.

Adottiamo una prassi per quando stiamo lavorando con componenti nuovi e che conosciamo poco: dai siti dei produttori o dei venditori è possibile scaricare i *datasheet*, ovvero i documenti con tutte le specifiche e i dettagli di funzionamento dei componenti. Imparare a leggerli e interpretarli è fondamentale per evitare problemi ed effettuare i collegamenti in modo corretto. Nella **Figura 4.5** è mostrato lo schema recuperato dal datasheet di un relè.

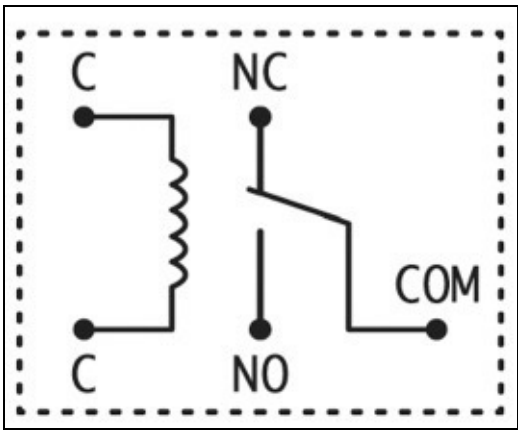


Figura 4.5 Lo schema elettrico di un relè tratto dal suo datasheet: è indispensabile per capire come effettuare correttamente i collegamenti.

In posizione di riposo il pin sulla destra COM (*common*) e quello al centro in alto NC (*normally closed*) sono collegati tra loro.

Facendo passare un impulso elettrico tra i due pin C sulla sinistra viene attivata la bobina (in inglese *coil*) rappresentata dalle linee curve nel diagramma: il pin NC sarà scollegato e verrà instaurata la connessione tra il pin COM e quello alla sua sinistra, NO (*normally open*).

Ricapitolando, l’elettricità è libera di fluire da COM a NC quando la bobina è in stato di riposo (non è applicato alcun voltaggio ai pin C), e da COM a NO quando la bobina è eccitata.

RICERCHE IN RETE

Utilizzando come chiave di ricerca le informazioni stampigliate sul componente, puoi recuperare in Rete il datasheet del modello di relè specifico che utilizzi. Leggine lo schema per dedurne la corretta modalità di connessione. Nel datasheet trovi anche tutti i dettagli sui carichi massimi di corrente che l’hardware può gestire in sicurezza.

Solitamente già sulla parte superiore del relè trovi riassunti i voltaggi, gli amperaggi limite del circuito principale e il voltaggio di riferimento per attivare la bobina.

Collegare il relè

Anche se alcuni tipi di relè possono essere collegati direttamente alla scheda Arduino, che può fornire una corrente adeguata per attivarne la bobina, è consigliabile utilizzare un transistor per evitare qualsiasi rischio di sovraccarico. Vediamo per prima cosa come realizzare il circuito.

NOTA *Lo sketch che abbiamo preparato negli esempi precedenti di questo capitolo lavora sul pin* `13`*, al quale è collegato il LED sulla scheda Arduino. Ti sarai accorto che ogni volta che colleghi l'alimentazione alla scheda o premi il tasto di reset il LED lampeggia alcune volte, e di conseguenza anche il relè verrebbe attivato e disattivato in rapida successione. È quindi consigliabile passare a un altro pin di output digitale: ti basta modificare il valore della variabile* `ledPin` *nella sezione iniziale dello sketch e realizzare il circuito utilizzando il pin scelto. Nella* Figura 4.6 *è stato scelto il pin* `4`*.*

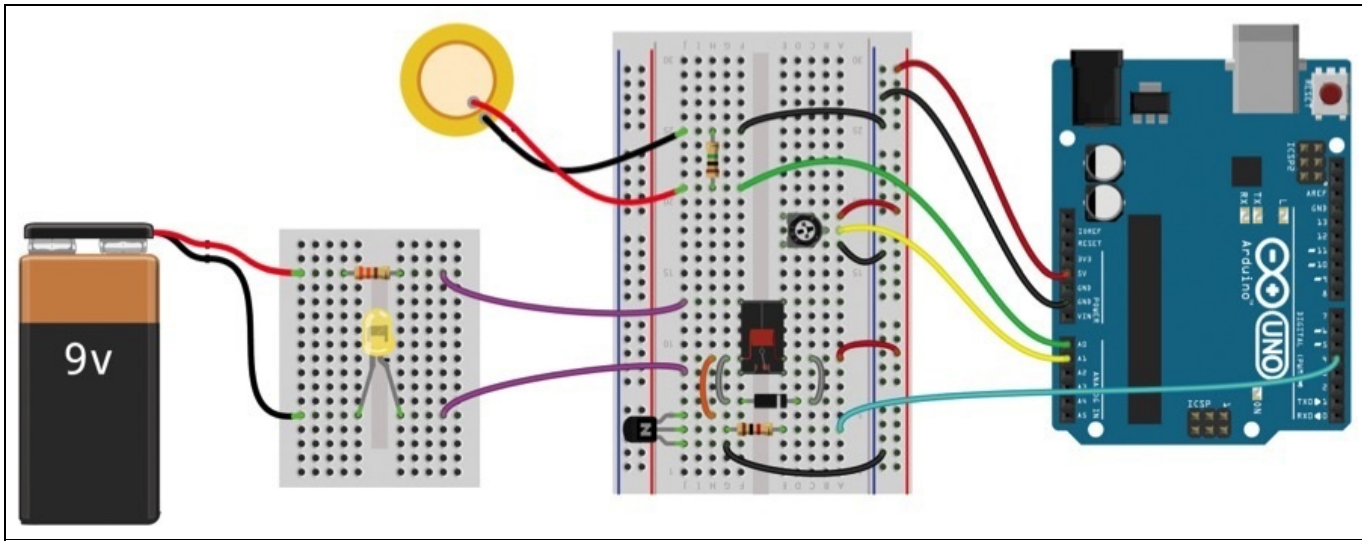


Figura 4.6 Sulla breadboard sono stati aggiunti il relè e i componenti necessari per il suo funzionamento; a fianco un esempio di circuito alimentato da una pila a 9V, elettricamente separato dalla scheda Arduino.

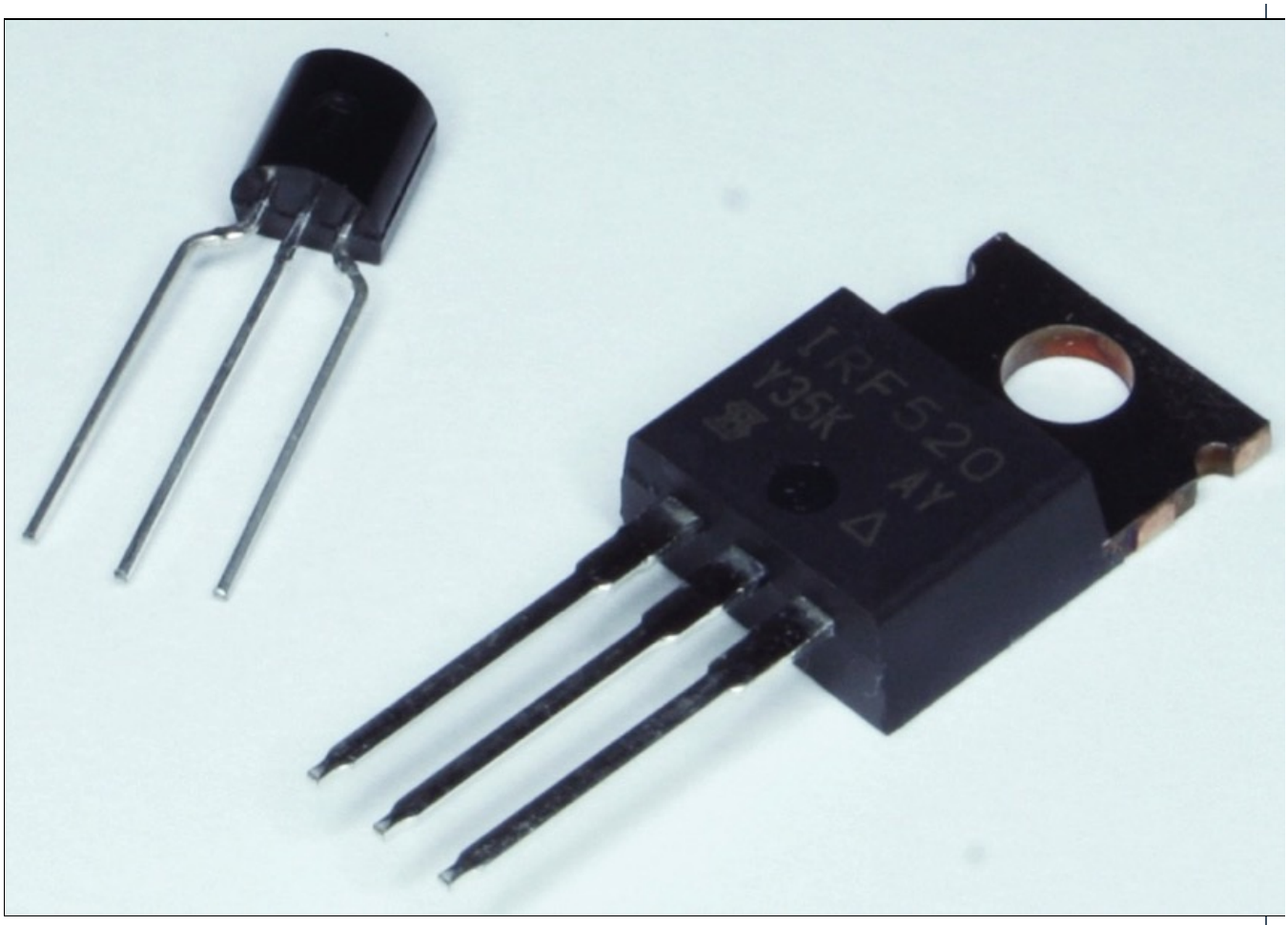
Il transistor (quello utilizzato in questo circuito è di tipo NPN) ha tre pin: quello centrale, definito *base*, è collegato al pin digitale 4: quando riceve una piccola quantità di corrente ne consente il passaggio tra gli altri due pin (*collettore* ed *emettitore*). Nel circuito della **Figura 4.6** il transistor si comporta di fatto come un interruttore, consentendo o meno il passaggio della corrente prelevata da 5V attraverso il relè verso GND.

L'attivazione del relè, a sua volta, chiude il circuito sulla sinistra, a cui è collegato con i due jumper viola: lì è presente una sorgente di alimentazione esterna e un utilizzatore di esempio. Potremo collegare utilizzatori ad assorbimento maggiore, fisicamente isolati (ma comunque comandati) dalla scheda Arduino. Sulla breadboard puoi notare anche un diodo collegato in parallelo al relè: il suo scopo è quello di impedire che il picco di corrente generato ogni volta che la bobina del relè viene disattivata possa danneggiare il transistor o la scheda. Un diodo permette il flusso di corrente solo in un senso, e nel caso specifico fa in modo che la corrente venga dissipata sulla bobina stessa e non sui componenti sensibili.

PERCHÉ UTILIZZARE UN TRANSISTOR

Il chip ATmega installato sulla scheda Arduino può gestire carichi di corrente fino a 40 mA su un singolo pin digitale e fino a 200 mA in totale. Oltre questi livelli il chip si danneggerà. Per comandare componenti con assorbimenti più elevati, come nel caso di alcuni relè, è necessario ricorrere all'utilizzo di transistor.

Esistono transistor differenti: fai sempre riferimento al datasheet per assicurarti di utilizzarne uno con caratteristiche elettriche adeguate.



Un transistor è in grado di regolare la resistenza opposta al passaggio di corrente tra collettore ed emettitore in relazione alla corrente applicata alla base. In questo modo è possibile ottenere comportamenti da interruttore o da amplificatore di segnale.

Seguendo a ritroso i collegamenti nella **Figura 4.6** ti renderai conto che l'alimentazione è fornita al relè attraverso il pin `5V`: questo pin, come gli altri della sezione `POWER`, non è gestito dal microcontrollore ed è perciò libero dai limiti di amperaggio dei pin digitali.

Conclusione

In questo capitolo abbiamo approfondito alcune problematiche legate all'acquisizione di input analogici: l'interpretazione dei valori, la taratura di un sensore e il successivo *fine tuning* (ritocco) dei dati letti per estrarre solo informazioni significative.

Abbiamo poi preso come spunto il circuito realizzato per affrontare un altro tema piuttosto importante: l'utilizzo di transistor e relè per comandare tramite la scheda Arduino carichi elettrici anche elevati mantenendoli elettricamente separati dalla scheda. Assicurati sempre di operare in sicurezza e di chiedere aiuto a un esperto se necessario quando lavori con tensioni e amperaggi elevati.

Arduino in Rete

Un tema che negli ultimi tempi sta riscuotendo un grande interesse è l'Internet delle Cose (Internet of Things, IoT): sempre più spesso oggetti intelligenti sono in grado di comunicare tra di loro e con la Rete, diventando più interattivi. Anche il mondo di Arduino ha preso spunto da questi temi, offrendo hardware dedicato e ispirando progetti davvero interessanti.

Il tema trattato in quest'ultimo capitolo sarà la realizzazione di una semplice stazione meteorologica: attraverso l'uso della rete cellulare, con lo shield GSM, la scheda sarà in grado di connettersi a Internet in modo indipendente per registrare temperatura, umidità e pressione atmosferica. Per prima cosa prenderemo confidenza con i sensori che utilizzeremo, più avanzati di quelli analizzati nei capitoli precedenti.

DHT11: temperatura e umidità

Il DHT11 è un sensore economico e affidabile, capace di restituire un segnale digitale calibrato con le informazioni lette. I range di lettura vanno dagli 0° ai 50° per la temperatura e dal 20% al 90% per l'umidità, con un'approssimazione rispettivamente di 2° e del 5%. Sono valori più che sufficienti per un utilizzo domestico.

> **NOTA** *Per letture più accurate puoi utilizzare l'analogo DHT22, con un range di lettura più ampio (da –40° a +80° per la temperatura e dal 5% al 99% per l'umidità) e una sensibilità maggiore (0,5° per la temperatura e 2% per l'umidità).*

La connessione del sensore è piuttosto semplice: da sinistra verso destra, il primo pin è collegato a `5V`, il secondo a un pin digitale per la lettura (nel nostro caso abbiamo scelto il pin `4`), il quarto a `GND`. Nota che non è necessario collegare il terzo pin. Per completare il collegamento, in parallelo tra `5V` e pin `4` collega un resistore da 10k Ohm.

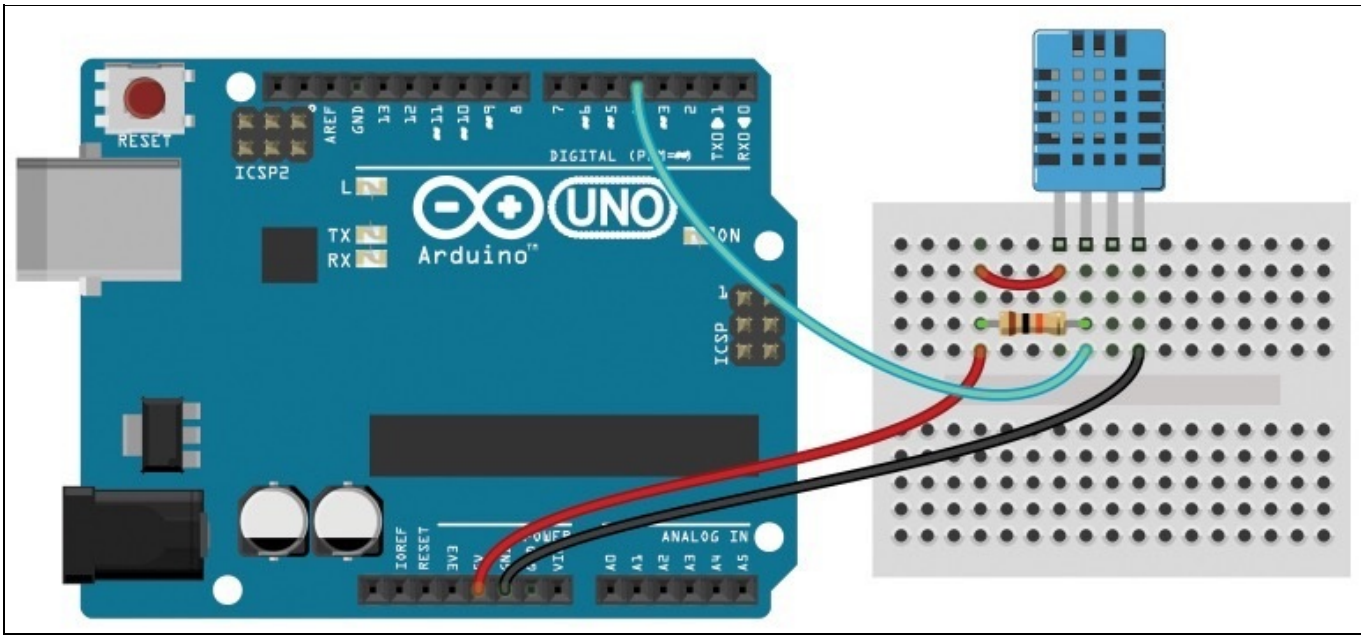


Figura 5.1 La connessione del sensore DHT11 alla scheda Arduino Uno.

> **NOTA** *In commercio puoi trovare questi sensori già montati su una piccola base dedicata, a cui devi collegare solo alimentazione,* `GND` *e segnale. In questo caso fai riferimento alle indicazioni del produttore per eseguire il collegamento nel modo corretto.*

I sensori digitali possono essere più evoluti di quelli analogici, poiché comunicano con il processore della scheda Arduino tramite sequenze di `0` e `1` (bit). Questo permette di poter avere a disposizione un segnale pulito, libero da interferenze e solitamente più affidabile. Di contro è necessario conoscere il protocollo di comunicazione adottato dal sensore in questione per poter interpretare i dati ricevuti.

Installare la libreria per il sensore DHT11

Molto spesso, soprattutto per i componenti più diffusi, sono disponibili librerie sviluppate e aggiornate da utenti stessi della community Arduino.

NOTA *Una libreria è un insieme di funzioni che, nel caso di un sensore, semplifica l'interpretazione dei dati trasmessi occupandosi "dietro le quinte" della comunicazione con l'hardware. A volte potrai trovare più librerie dedicate allo stesso componente. In questi casi confronta le caratteristiche e le peculiarità di ognuna, esegui dei test, e tieni presente anche quanto sono recenti.*

Una rapida ricerca in Rete (usando come criterio la sigla del componente e "libreria Arduino") ci porta sulla pagina di GitHub `https://github.com/adafruit/DHT-sensor-library`, da cui è possibile scaricare una libreria che fa proprio al caso nostro. Con un clic sul pulsante **Download ZIP**, sulla destra, avviamo il processo.

Una volta terminato il download, avvia l'IDE Arduino e seleziona **Aggiungi libreria** (o **Add Library**) dal menu **Sketch** > **Importa libreria**. Naviga fino alla posizione dell'archivio ZIP scaricato e fai clic su **Scegli**.

NOTA *Prima dell'importazione è consigliabile rinominare lo ZIP eliminando eventuali – dal nome: potrebbero interferire con il processo di importazione.*

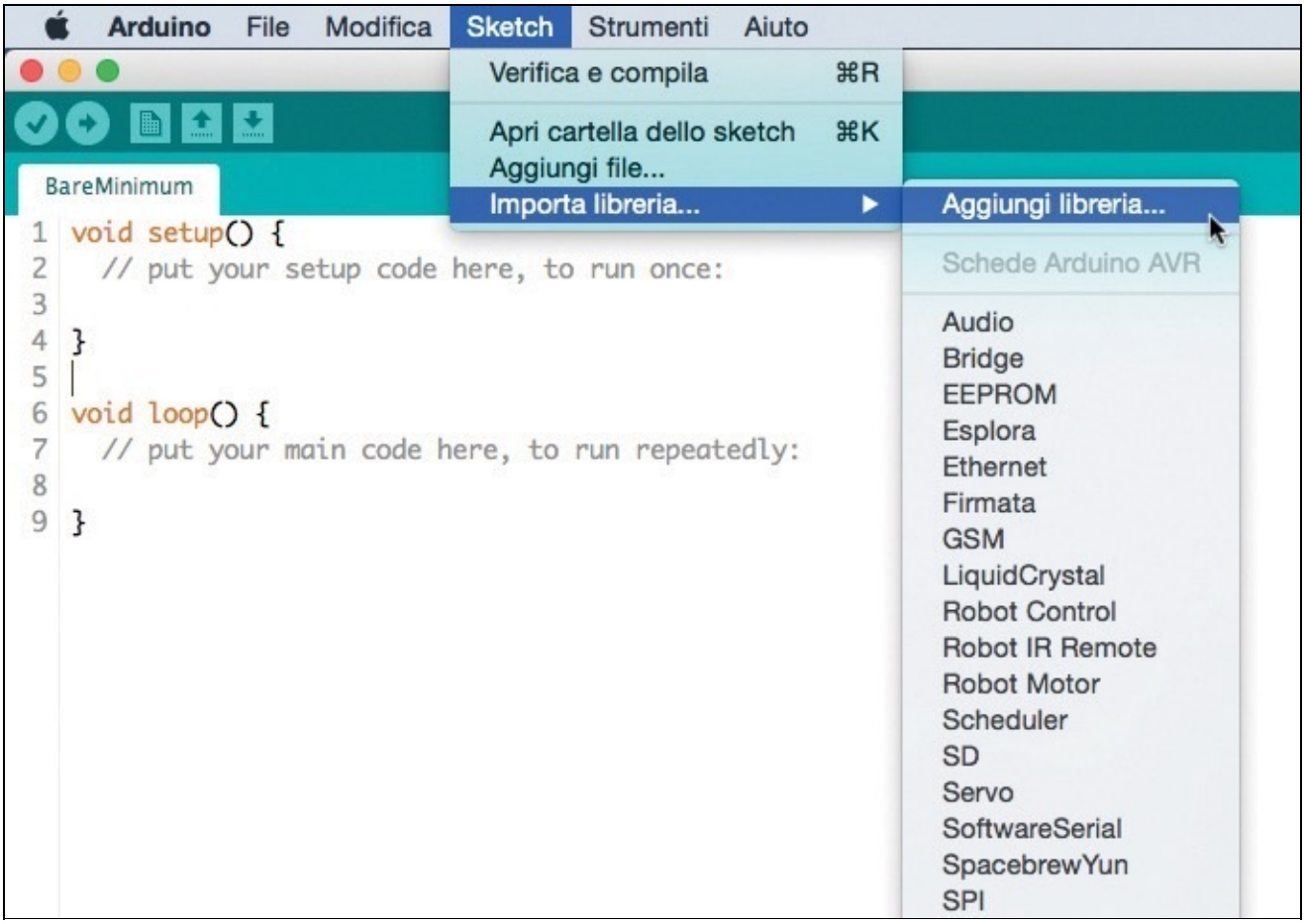


Figura 5.2 La voce Aggiungi libreria consente di importare librerie esterne da affiancare a quelle predefinite.

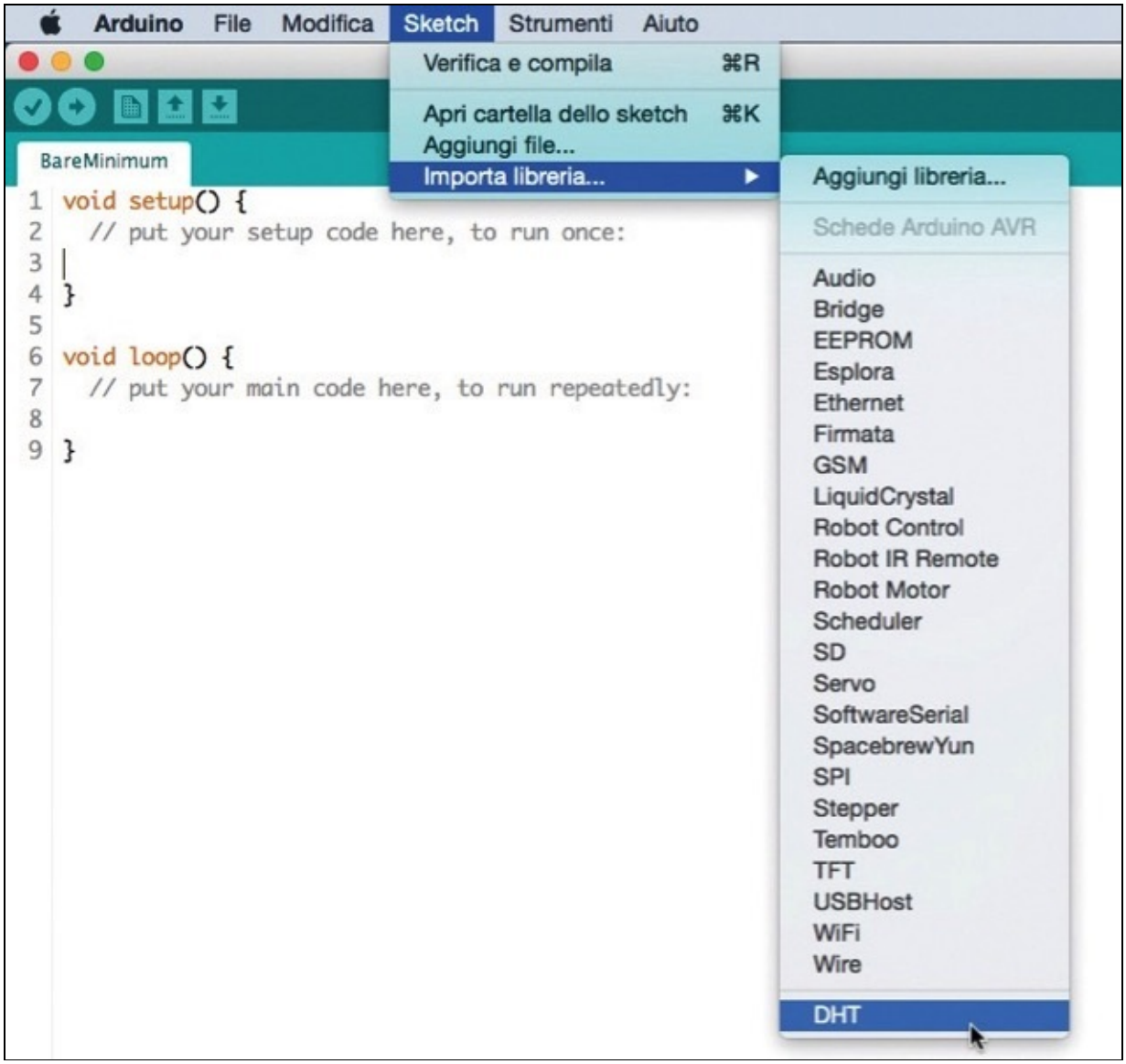


Figura 5.3 Dopo l'importazione il menu Importa libreria ha una nuova voce.

IMPORTAZIONE MANUALE DELLA LIBRERIA

Oltre al sistema automatico di importazione appena descritto, puoi rendere disponibile una libreria nell'IDE anche estraendo il contenuto dell'archivio ZIP nella cartella **libraries** che trovi nello sketchbook (una sottocartella per ogni libreria): la posizione standard dello sketchbook è nella cartella **Documenti/Arduino** dell'utente che utilizzi.

Tieni presente che per avere disponibili nell'IDE le librerie importate in questo modo dovrai riavviare l'applicazione.

La maggior parte delle librerie viene distribuita completa di alcuni sketch di esempio, adatti per familiarizzare con le funzioni messe a disposizione. Trovi gli esempi nel sottomenu dedicato a ogni specifica libreria facendo clic sul pulsante **Apri** nella barra degli strumenti. Nel caso della libreria appena installata, è presente un solo sketch di esempio, **DHTtester**, ricco di commenti, utilissimo per eseguire un test preliminare con il sensore. Cerchiamo di analizzare le parti più importanti del codice.

La prima istruzione comunica al compilatore che nello sketch verranno utilizzate funzioni presenti nella libreria DHT:

```
#include "DHT.h"
```

La seconda istruzione è necessaria per impostare il pin a cui è collegato il sensore. Nel nostro caso, per evitare interferenze con lo shield GSM, modificheremo fin d'ora il pin, impostando il numero `4`:

```
#define DHTPIN 4
```

Il successivo blocco di istruzioni permette di selezionare il tipo di sensore con cui stiamo comunicando; dobbiamo lasciare non commentata solo la definizione adatta, nel nostro caso quella relativa al componente DHT11:

```
#define DHTTYPE DHT11     // DHT 11
// #define DHTTYPE DHT22  // DHT 22 (AM2302)
// #define DHTTYPE DHT21  // DHT 21 (AM2301)
```

Segue l'inizializzazione del sensore secondo i parametri appena impostati:

```
DHT dht(DHTPIN, DHTTYPE);
```

A questo punto la funzione `setup()` attiva la comunicazione seriale e il sensore DHT:

```
void setup() {
  Serial.begin(9600);
  Serial.println("DHTxx test!");
  dht.begin();
}
```

Nella funzione `loop()`, non mostrata qui integralmente per brevità, vengono eseguite le letture e successivamente comunicate al monitor seriale. In particolare, nota l'utilizzo delle istruzioni `dht.readHumidity()` e `dht.readTemperature()` specifiche per il sensore.

Trasferiamo lo sketch e apriamo il monitor seriale per osservare i valori letti dal sensore.

Possiamo vedere i dati relativi a umidità, temperatura (in gradi Celsius e Fahrenheit) e indice di calore percepito.

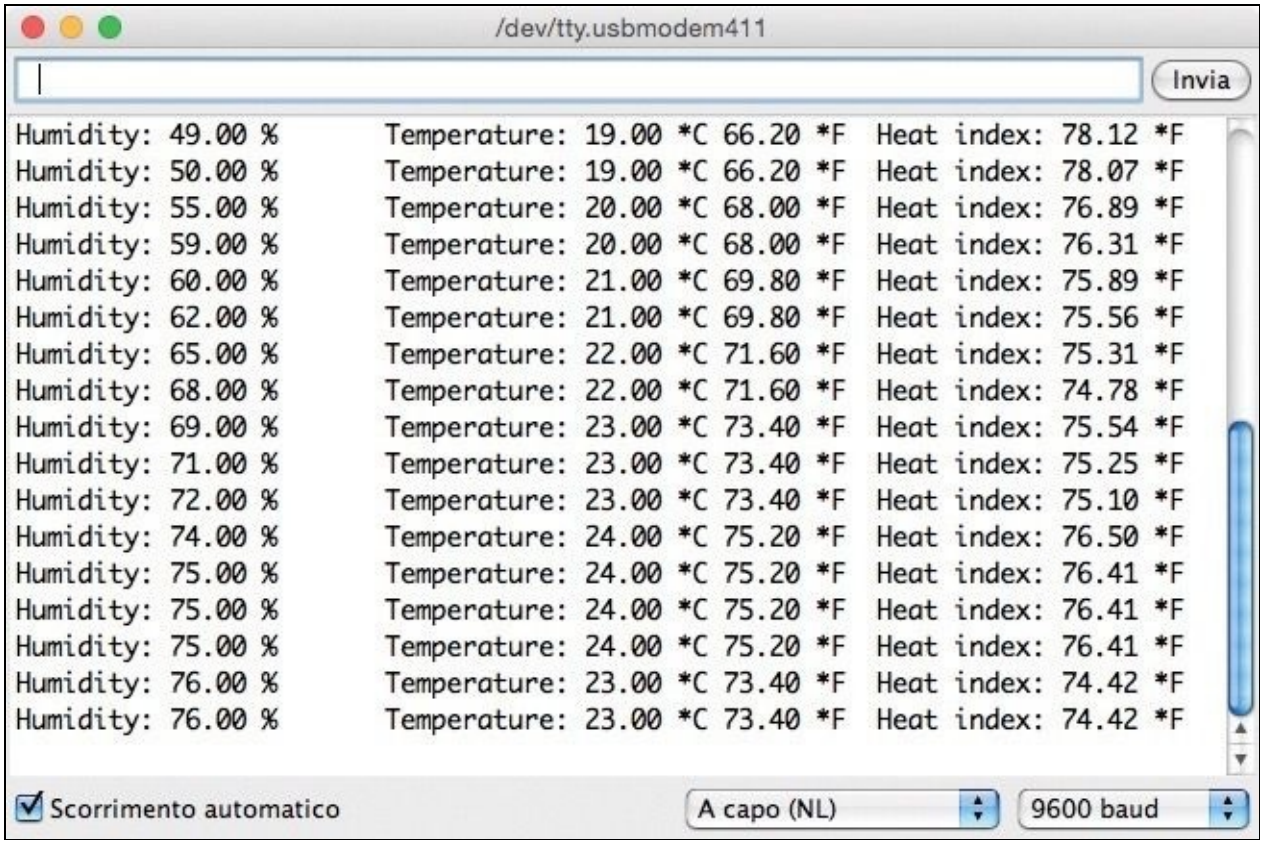


Figura 5.4 Il monitor seriale con i dati comunicati dal sensore DHT11.

Con poche modifiche possiamo estrapolare solo i dati che ci interessano, nel nostro caso umidità e temperatura in gradi Celsius:

```
void loop() {
  // Attesa di due secondi tra una lettura e la successiva
  delay(2000);

  // La lettura richiede circa 250 millisecondi
  // Inoltre i dati possono essere aggiornati
  // a due secondi fa (è un sensore molto lento)
  float h = dht.readHumidity();
  // Lettura della temperatura
  float t = dht.readTemperature();
  // Verifica se la lettura è andata a buon fine
  if (isnan(h) || isnan(t)) {
    Serial.println("Errore di lettura dal sensore DHT");
  return;
  }
  Serial.print("Um: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temp: ");
  Serial.println(t);
}
```

A questo punto, sicuri del funzionamento di questo sensore, passiamo ad analizzare il funzionamento del sensore barometrico.

BMP180: temperatura e pressione atmosferica

Il BMP180 è un sensore molto preciso, anche in questo caso digitale, capace di leggere le variazioni della pressione atmosferica. L'estrema precisione e affidabilità ne consentono, con le dovute tarature, anche l'uso come altimetro (con un'accuratezza nell'ordine di un metro).

La comunicazione con questo sensore si basa sull'interfaccia standard *I2C*, definita anche *TWI* o *Wire*, un protocollo disponibile nativamente su molti microcontrollori. Questa connessione richiede l'utilizzo di soli due pin, uno definito *Clock* (che si occupa della sincronizzazione dei messaggi inviati e ricevuti) e l'altro *Data* (attraverso il quale vengono effettivamente trasmesse le informazioni). Uno dei vantaggi della comunicazione I2C è la possibilità di collegare più dispositivi I2C allo stesso *bus* (la coppia di cavi), senza utilizzare più pin del microcontrollore.

Nel caso della scheda Arduino Uno, i pin riservati a questo tipo di comunicazione sono A5 (*Clock*) e A4 (*Data*).

> **NOTA** *Dalla revisione R3 della scheda, trovi due pin dedicati,* SCL *(*Clock*) e* SDA *(*Data*), a fianco del pin* AREF*.*

Il sensore BMP180 è quasi sempre disponibile su una scheda di *breakout* dedicata, che raccoglie tutti i componenti necessari per il suo corretto funzionamento e ne semplifica la connessione. Verifica i dettagli di quella in tuo possesso per eseguire i collegamenti nel modo giusto.

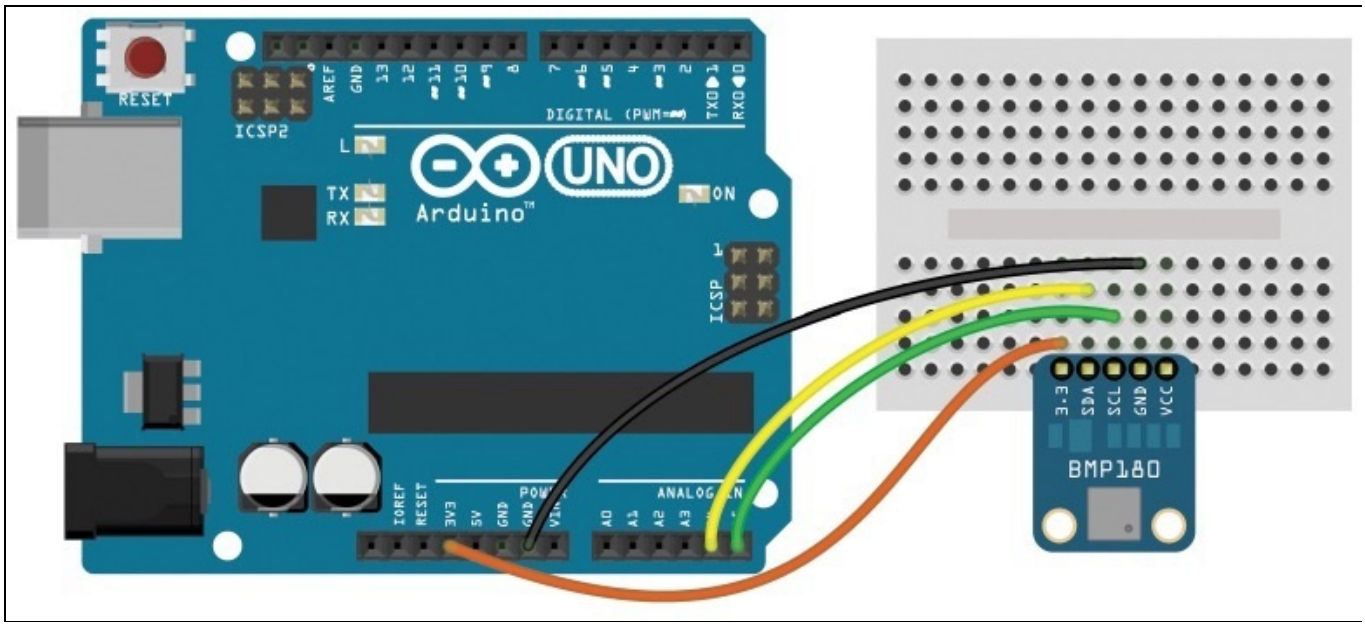


Figura 5.5 Il sensore BMP180 collegato all'interfaccia I2C.

NOTA *Fai attenzione: per non danneggiare il sensore, assicurati sempre di alimentarlo con un voltaggio di 3.3 Volt. Se la scheda di breakout in tuo possesso non si occupa della conversione, alimentala tramite il pin* `3.3V` *già presente sulla scheda Arduino.*

Installare la libreria per il sensore BMP180

In modo analogo a quanto accaduto per il sensore DHT11, una rapida ricerca in Rete ci porta a individuare su GitHub la pagina di una libreria dedicata proprio alla famiglia di sensori BMP (`https://github.com/sparkfun/BMP180_Breakout`). L'installazione del pacchetto è analoga: scarica l'archivio ZIP e segui la procedura dal menu **Sketch** > **Importa libreria** > **Aggiungi Libreria**.

A questo punto apriamo lo sketch di esempio **SFE_BMP180_example** dalla nuova voce nel menu **Apri** della barra degli strumenti.

Dopo il commento iniziale con i dettagli su connessione del sensore e utilizzo, due istruzioni includono le librerie necessarie: oltre alla libreria specifica per il sensore BMP180, è inclusa anche la libreria Wire, già disponibile nell'IDE Arduino, che si occupa della comunicazione I2C:

```
#include <SFE_BMP180.h>
#include <Wire.h>
```

Viene poi inizializzato l'oggetto `pressure` legato alla libreria…

```
SFE_BMP180 pressure;
```

… e definita la costante `ALTITUDE` relativa alla quota di partenza delle misurazioni:

```
#define ALTITUDE 125.0
```

Nella funzione `setup()` viene eseguita una verifica sul corretto funzionamento del sensore, utile per scongiurare eventuali errori nella connessione:

```
void setup() {
  Serial.begin(9600);
  Serial.println("REBOOT");

  // Inizializzazione del sensore (importante
  // per avere valori di calibrazione).
  if (pressure.begin()) {
    Serial.println("Inizializzazione BMP180 OK");
  } else {
    // Verificare connessioni
    Serial.println("Inizializzazione BMP180 NO");
    while(1); // Pausa infinita.
  }
}
```

Nella funzione `loop()` vengono eseguite le letture e successivamente comunicate al monitor seriale. Anche in questo caso è piuttosto semplice intuire la funzione dei comandi forniti dalla libreria: `startTemperature()` e `startPressure()` avviano

rispettivamente la lettura della temperatura e della pressione, restituendo il tempo necessario per la lettura stessa, valore che viene utilizzato come parametro di `delay()` per mettere in pausa lo sketch per il tempo necessario; `getTemperature()` e `getPressure()` interpretano il dato letto. In particolare, si può notare che la funzione `getPressure()` richiede come parametro anche la temperatura letta per garantire un valore più preciso possibile eseguendo i necessari aggiustamenti della lettura.

> **NOTA** *Quando viene installata una nuova libreria, nell'IDE le keyword relative alle nuove funzioni disponibili vengono colorate in arancione: è un aiuto in più che ti permetterà di evitare errori di digitazione.*

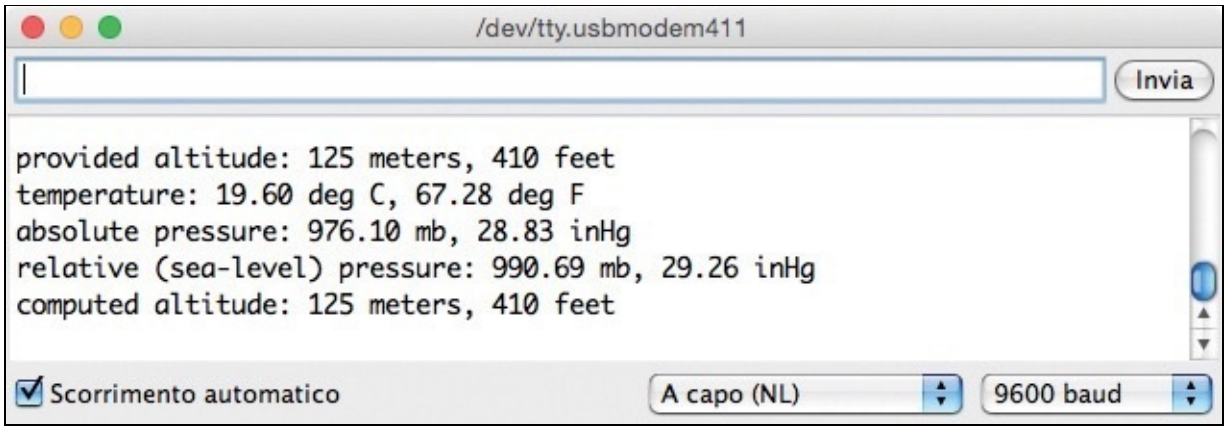


Figura 5.6 Il monitor seriale con i dati comunicati dal sensore BMP180.

Come per il sensore analizzato in precedenza, modifichiamo il loop dello sketch di esempio per visualizzare solo i valori a cui siamo interessati: pressione al livello del mare e temperatura:

```
void loop() {
 char status;
 double T, P, p0, a;
 // Prima di eseguire la lettura della pressione
 // è necessario ottenere la temperatura.
 // Avvia la misura della temperatura:
 // se la richiesta ha successo, viene restituito
 // il numero di millisecondi di attesa.
 // In caso contrario viene restituito 0.

 status = pressure.startTemperature();
 if (status != 0) {
  // Attendi che la lettura sia completa:
  delay(status);
  // Recupera la lettura della temperatura:
  // viene salvata in T.
  // La funzione restituisce 1 se ok,
  // 0 in caso di problemi.

  status = pressure.getTemperature(T);
  if (status != 0) {
   // Comunica temperatura:
   Serial.print("temperatura: ");
   Serial.print(T, 2);
   // Avvia la misura della pressione:
   // Il parametro indica la risoluzione, da 0 a 3
   // (più alto, più lento).
   // Se la richiesta ha successo, viene restituito
   // il numero di millisecondi di attesa.
   // In caso contrario viene restituito 0.
```

```
   status = pressure.startPressure(3);
   if (status != 0) {
    // Attendi che la lettura sia completa:
    delay(status);
    // Recupera la lettura della pressione:
    // viene salvata in P.
    // Tra i parametri è presente la temperatura
    // letta in precedenza (T).
    // La funzione restituisce 1 se ok,
    // 0 in caso di problemi.

    status = pressure.getPressure(P, T);
    if (status != 0) {
     // Comunica pressione compensata
     // sul livello del mare:
     p0 = pressure.sealevel(P, ALTITUDE);
     Serial.print(" pressione: ");
     Serial.print(p0, 2);
     Serial.println(" mb");
    } else {
     Serial.println("errore durante recupero
     pressione");
    }
   } else {
    Serial.println("errore durante inizio lettura
    pressione");
   }
  } else {
   Serial.println("errore durante recupero
   temperatura");
  }
 } else {
  Serial.println("errore durante inizio lettura
  temperatura");
 }
 delay(5000);  // Pausa per 5 secondi
}
```

Le condizioni `if()` nidificate assicurano la lettura sequenziale della temperatura prima, e della pressione poi. In caso di errore viene inviato un messaggio specifico.

Come vedi, la logica operativa è in parte differente rispetto a quella della libreria analizzata in precedenza: non esiste un vero e proprio standard, ma solitamente è sufficiente un’analisi approfondita degli sketch di esempio per comprendere il funzionamento di ognuna.

Combinare i due sensori

Dopo aver compreso come lavorano i due sensori, possiamo realizzare uno sketch per ottenere la lettura dei dati a cui siamo interessati: è l'ultimo passo prima di occuparci della connessione in Rete.Dal punto di vista delle connessioni non cambia nulla: i due sensori restano indipendenti tra loro e ognuno è collegato ai pin necessari per il suo funzionamento.

NOTA *Qualora avessi a che fare con più sensori che lavorano allo stesso voltaggio, il più delle volte può essere comodo utilizzare le piste di alimentazione laterali messe a disposizione dalla maggior parte delle breadboard.*

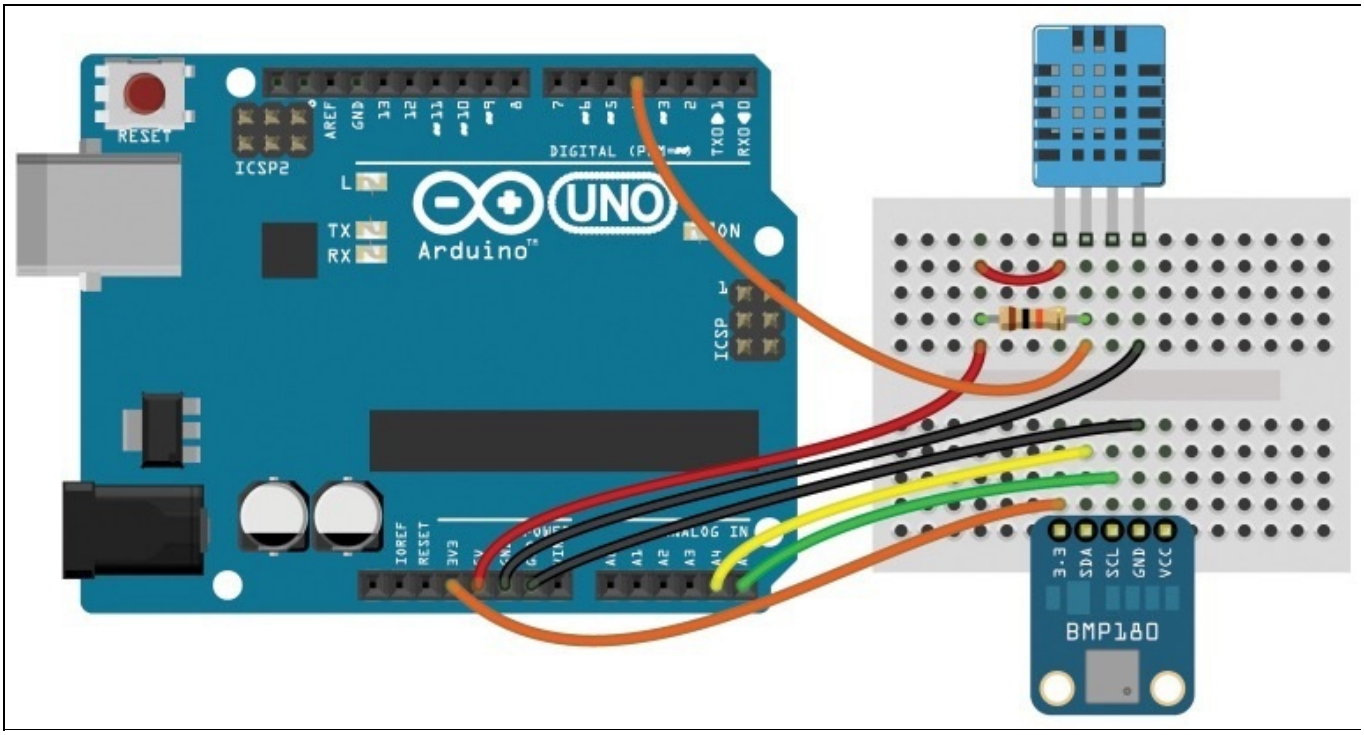


Figura 5.7 I due sensori collegati alla scheda Arduino Uno: la stazione meteo è quasi completa.

Organizziamo lo sketch, combinando quelli di esempio appena descritti, mantenendo la stessa logica. Nella prima parte è indispensabile importare le librerie necessarie per comunicare con i due sensori:

```
// importazione delle librerie
#include <DHT.h>
#include <SFE_BMP180.h>
#include <Wire.h>
```

Subito dopo dichiariamo le costanti che definiscono i parametri necessari al corretto funzionamento dell'hardware:

```
// definizione pin per il sensore DHTxx
#define DHTPIN 4
// definizione del tipo di sensore DHTxx
#define DHTTYPE DHT11  // scegli tra DHT11 DHT21 DHT22
// definizione della quota delle misurazioni
#define QUOTA 125.0 // quota di Milano, in metri
```

Non resta che inizializzare le istanze dei due sensori, facendo riferimento alle istruzioni `DHT` e `SFE_BMP180` definite nelle librerie:

```
// inizializzazione sensore DHTxx
DHT dht(DHTPIN, DHTTYPE);
// inizializzazione sensore BMPxxx
SFE_BMP180 bmp;
```

La funzione `setup()` apre la comunicazione seriale, che utilizzeremo per visualizzare le letture, e inizializza il sensore BMP180 con `bmp.begin()`:

```
void setup() {
  Serial.begin(9600);
  dht.begin();
  if (!bmp.begin()) {
    Serial.println("Errore BMPxxx");
    while(1); // Pausa in caso di errore
  }
}
```

NOTA *Il punto esclamativo (*`!`*) nella condizione* `if()` *è una negazione: quando a causa di un problema (per esempio di connessione) la funzione* `bmp.begin()` *non si conclude in modo corretto, viene trasmesso al monitor seriale un messaggio di errore e lo sketch viene interrotto con un ciclo infinito* `while(1)`*.*

Apriamo la funzione `loop()` con un `delay()` piuttosto importante; i dati che stiamo leggendo non variano istantaneamente, perciò per i test preliminari una lettura ogni dieci secondi è più che sufficiente:

```
void loop() {
  // una lettura ogni 10 secondi
  delay(10000);
```

Segue un blocco con le dichiarazioni delle variabili che utilizzeremo nello sketch. Come sempre il nome è *case sensitive*, perciò è necessario prestare attenzione a maiuscole e minuscole:

```
// variabili per il sensore DHTxx
float DHTt, DHTh;
// variabili per il sensore BMPxxx
char stato;
double BMPt,BMPp,BMPp0;
```

LA VISIBILITÀ DELLE VARIABILI

A differenza di quanto fatto in molti altri sketch di esempio, in questo caso abbiamo scelto di dichiarare le variabili direttamente nel loop, anziché nella sezione iniziale dello sketch: in gergo lo *scope* di queste variabili è la funzione `loop()`.

Le variabili dichiarate all'interno di una funzione sono visibili e utilizzabili solo all'interno della stessa (e delle funzioni in essa nidificate), mentre non sono disponibili all'esterno: per esempio, una variabile dichiarata in `setup()` non sarà visibile in `loop()`.

Tieni anche presente che le variabili così dichiarate verranno inizializzate ogni volta che la funzione sarà richiamata, e di conseguenza non conserveranno l'eventuale valore acquisito in precedenza.

È la volta delle istruzioni dedicate alla lettura del sensore DHT11. Con le funzioni specifiche `readTemperature()` e `readHumidity()` viene attribuito un valore rispettivamente alle variabili `DHTt` e `DHTh`. Un controllo tramite la funzione `isnan` (letteralmente *is not a number*, ovvero “non è un numero”) garantisce che i dati siano stati recuperati correttamente:

```
// lettura temperatura dal sensore DHTxx
DHTt = dht.readTemperature();
// lettura umidità dal sensore DHTxx
DHTh = dht.readHumidity();
// verifica dei dati letti
if (isnan(DHTt) || isnan(DHTh)) {
  Serial.println("Errore lettura DHTxx");
  return; // riavvia il loop in caso di errore
}
```

In modo analogo, seguendo il codice dello sketch di esempio della libreria BMP180, vengono rilevate temperatura e pressione. Le condizioni `if()` nidificate assicurano la lettura della temperatura, indispensabile per ottenere un valore della pressione corretto. In caso contrario il valore `0` della variabile `stato` identifica un problema e interrompe il `loop()`:

```
// lettura temperatura e pressione da BMPxxx
stato = bmp.startTemperature();
if (stato != 0) {
  delay(stato);
  stato = bmp.getTemperature(BMPt);
  if (stato != 0) {
    stato = bmp.startPressure(3); // precisione da 0 a 3
    if (stato != 0) {
      delay(stato);
      stato = bmp.getPressure(BMPp,BMPt);
      if (stato != 0) {
        BMPp0 = bmp.sealevel(BMPp,QUOTA);
      }
    }
  }
}
// verifica dei dati letti
if (stato == 0) {
  Serial.println("Errore lettura BMPxxx");
  return; // riavvia il loop in caso di errore
}
```

Non resta che comunicare i dati attraverso la connessione seriale prima di concludere il `loop()`:

```
  // comunicazione delle letture
  Serial.print("Temp (C): ");
  Serial.print(DHTt,2);
  Serial.print("\t Um (%): ");
  Serial.print(DHTh,2);
  Serial.print("\t Temp (C): ");
  Serial.print(BMPt,2);
  Serial.print("\t Press (mbar): ");
  Serial.println(BMPp,2);
}
```

Trasferisci lo sketch sulla scheda e apri il monitor seriale per verificare l’output.

NOTA *Il sensore di temperatura associato al BMP180 è più preciso di quello contenuto nel DHT11: lo puoi verificare facilmente confrontando le letture affiancate nel monitor seriale.*

La connessione in Rete: lo shield GSM e Plot.ly

Lo shield GSM, sviluppato dal team Arduino in collaborazione con Telefónica, è pensato appositamente per dotare la scheda di connettività grazie alla rete GSM.

Con questo shield installato è possibile navigare in Rete, inviare e ricevere SMS e, con poche modifiche all'hardware, effettuare e ricevere telefonate.

> **NOTA** *Nella confezione dello shield è inclusa una scheda SIM esclusiva, che offre un piano tariffario pensato appositamente per l'uso tipico della connessione in Rete con Arduino (brevi pacchetti di dati), ma tieni presente che puoi inserire nello slot qualsiasi SIM valida sul network GSM. In particolare, ricorda che la SIM inclusa è abilitata solo per la connessione in Rete: non hai la possibilità di ricevere o inviare SMS e telefonate. Se per il tuo progetto hai necessità di utilizzare queste funzioni, procurati una SIM adatta.*

L'impiego dello shield è molto semplice: l'IDE Arduino mette a disposizione una libreria dedicata, con molti sketch di esempio utili sia per i test iniziali, sia per comprendere appieno il funzionamento dell'hardware.

Per quanto riguarda il collegamento dei sensori, i pin della scheda principale sono riportati sullo shield: non devi far altro che trasferire i jumper uno per uno.

Lo shield GSM comunica con la scheda Arduino Uno su cui è installato tramite i pin 2 e 3. Inoltre il pin 7 è riservato per il reset del modem. Quando colleghi altri componenti, assicurati di non utilizzare tali pin.

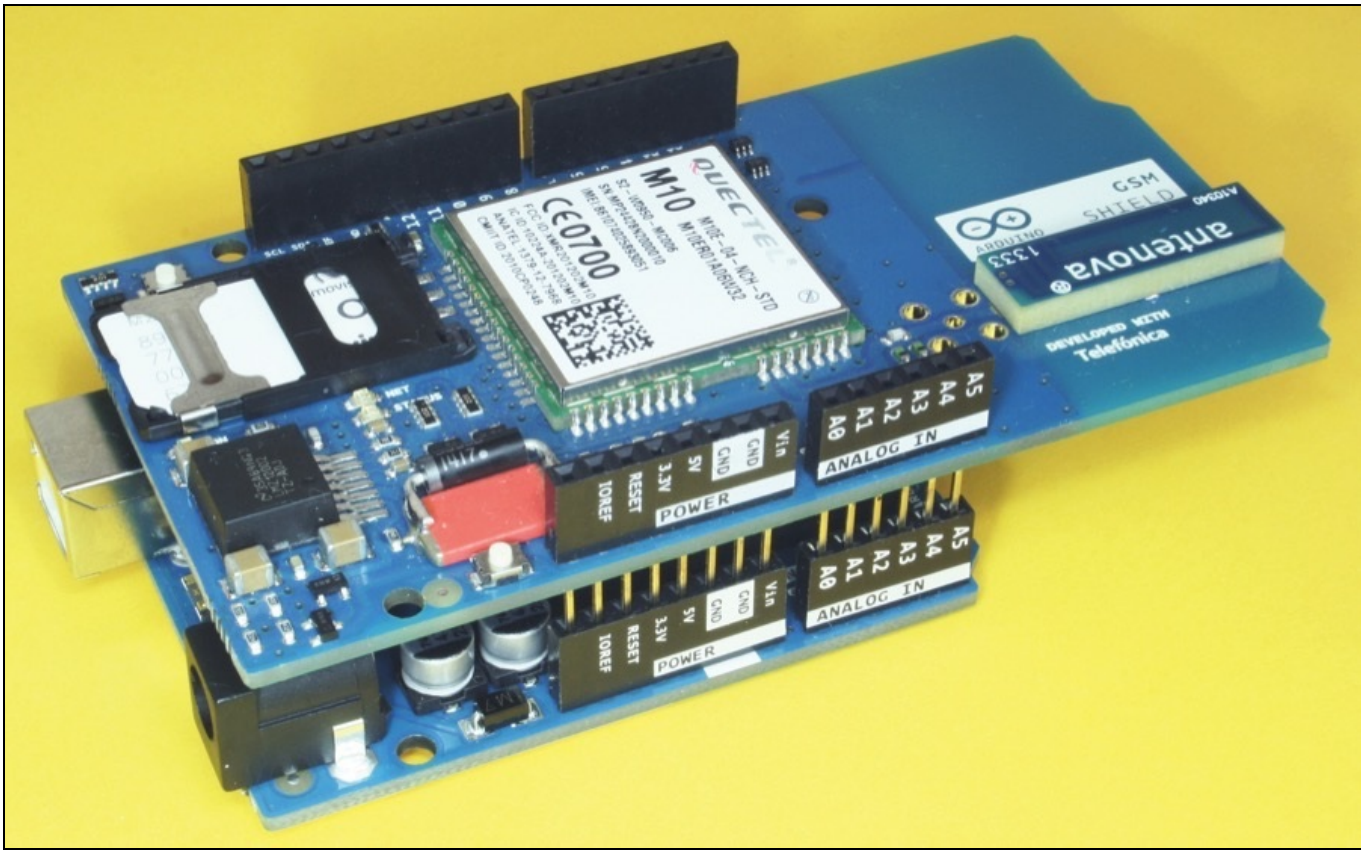
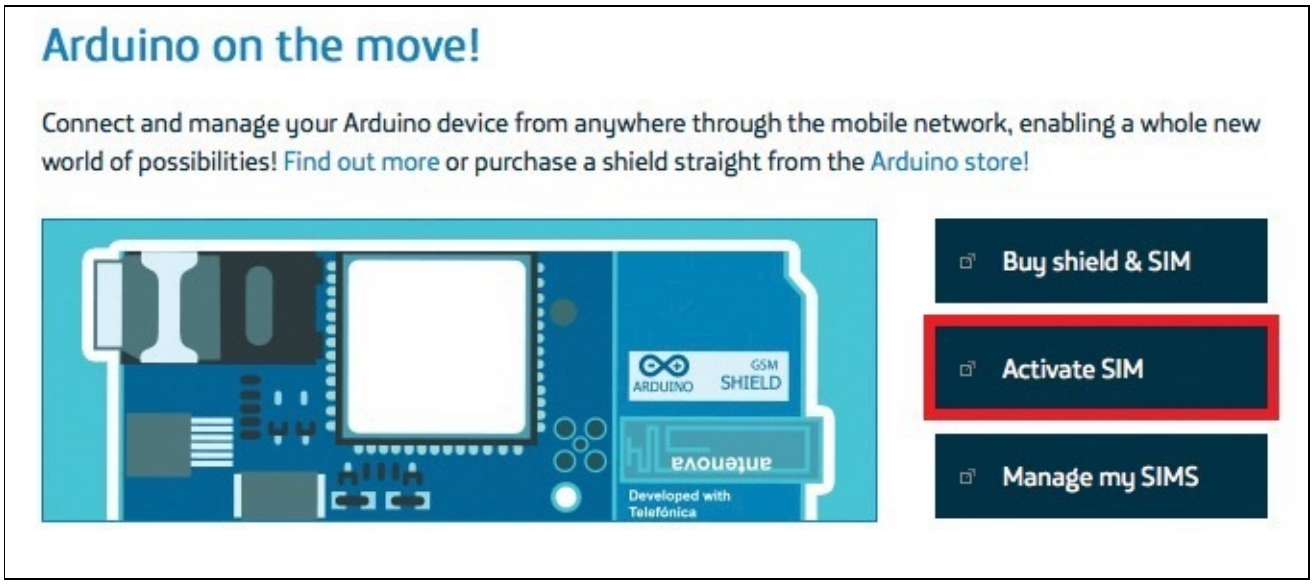


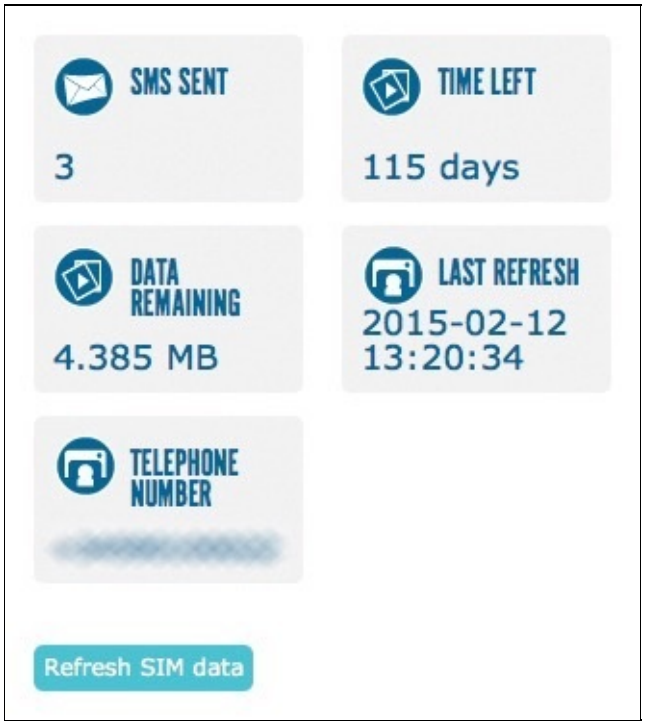
Figura 5.8 Lo shield GSM montato sulla scheda Arduino Uno.

ATTIVA LA SIM TELEFÓNICA

Per attivare la SIM inclusa nella confezione vai all'indirizzo dedicato `http://arduino.movilforum.com`, fai clic su **Activate SIM** e registrati.



Inserisci il numero della tua SIM (costituito dalle 19 cifre stampigliate su di essa), seleziona il pacchetto dati ed esegui il pagamento tramite PayPal. Ad attivazione avvenuta, nella sezione **Manage My SIM cards** fai clic sul nome assegnato alla scheda per visualizzare i relativi dettagli, inclusi il credito residuo, la validità della SIM e le statistiche d'uso.



Un’ultima nota: utilizzando questa SIM, negli sketch dovrai digitare come APN `sm2ms.movilforum.es`, senza indicare alcun nome utente né password.

I test di connessione preliminari

Prima di tutto è necessario verificare che la connessione alla rete GSM sia attiva. Per farlo apriamo e trasferiamo sulla scheda lo sketch **GsmScanNetworks** presente tra gli esempi nel menu **File** > **Esempi** > **GSM** > **Tools**.

Completato il caricamento, apriamo il monitor seriale e restiamo in attesa dell’output (il processo potrebbe richiedere alcuni minuti). Sullo shield, tre LED ti informano sullo stato dell’alimentazione (ON fisso), della connessione (STATUS fisso) e della comunicazione (NET lampeggiante).

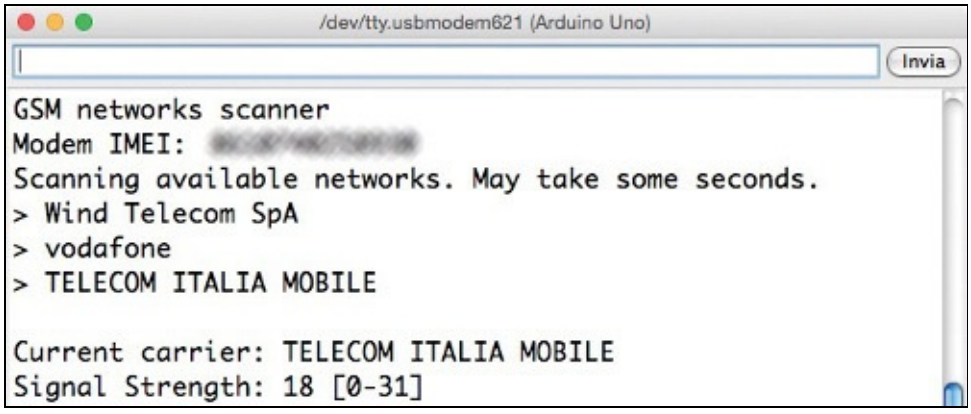


Figura 5.9 L’output di GsmScanNetworks mostra la rete a cui è connesso lo shield.

Un’ulteriore possibile verifica riguarda la connessione a Internet: scegli lo sketch **GsmWebClient** da **File** > **Esempi** > **GSM**. In questo caso prima di caricarlo sulla scheda è necessario impostare i corretti parametri di connessione nella sezione iniziale dello sketch:

```
// PIN Number
#define PINNUMBER ""

// APN data
#define GPRS_APN      "GPRS_APN"
// replace your GPRS APN
#define GPRS_LOGIN    "login"
// replace with your GPRS login
#define GPRS_PASSWORD  "password"
// replace with your GPRS password
```

Vengono richiesti il codice PIN (`PINNUMBER`) associato alla scheda SIM (le schede Telefónica vengono fornite senza PIN), l’APN `GPRS_APN`, il login `GPRS_LOGIN` e la password `GPRS_PASSWORD`, da digitare tra le virgolette `"`. Verifica con il gestore della tua SIM i dati da inserire.

A questo punto puoi caricare lo sketch e aprire il monitor seriale per attendere l’output.

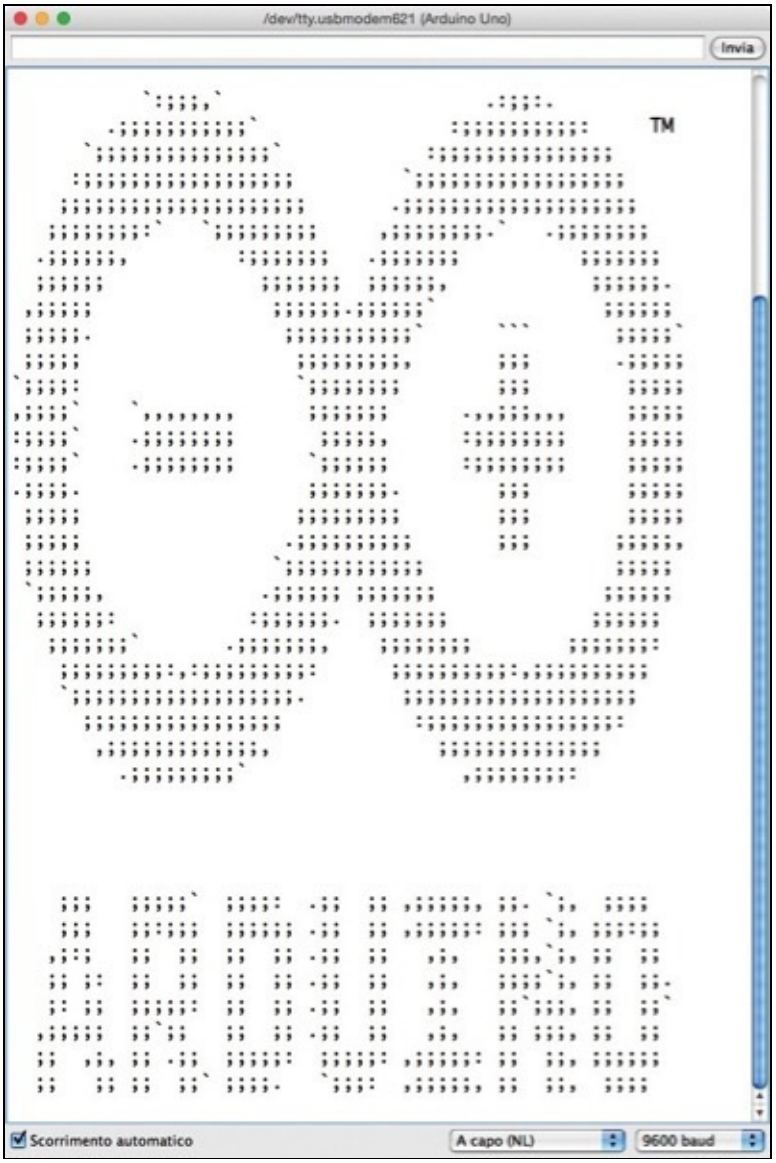


Figura 5.10 La risposta ricevuta dal server con lo sketch GsmWebClient di esempio.

I tuoi grafici online: Plot.ly

Per visualizzare l’andamento delle letture della stazione meteorologica che stiamo creando ci affideremo a un servizio online dedicato alla realizzazione di grafici e diagrammi. Inviando i valori letti dai sensori potremo ricostruire l’andamento di temperatura, pressione e umidità, per consultare i dati in ogni momento. Vai all’indirizzo `https://plot.ly` e crea un account scegliendo nome utente e password.

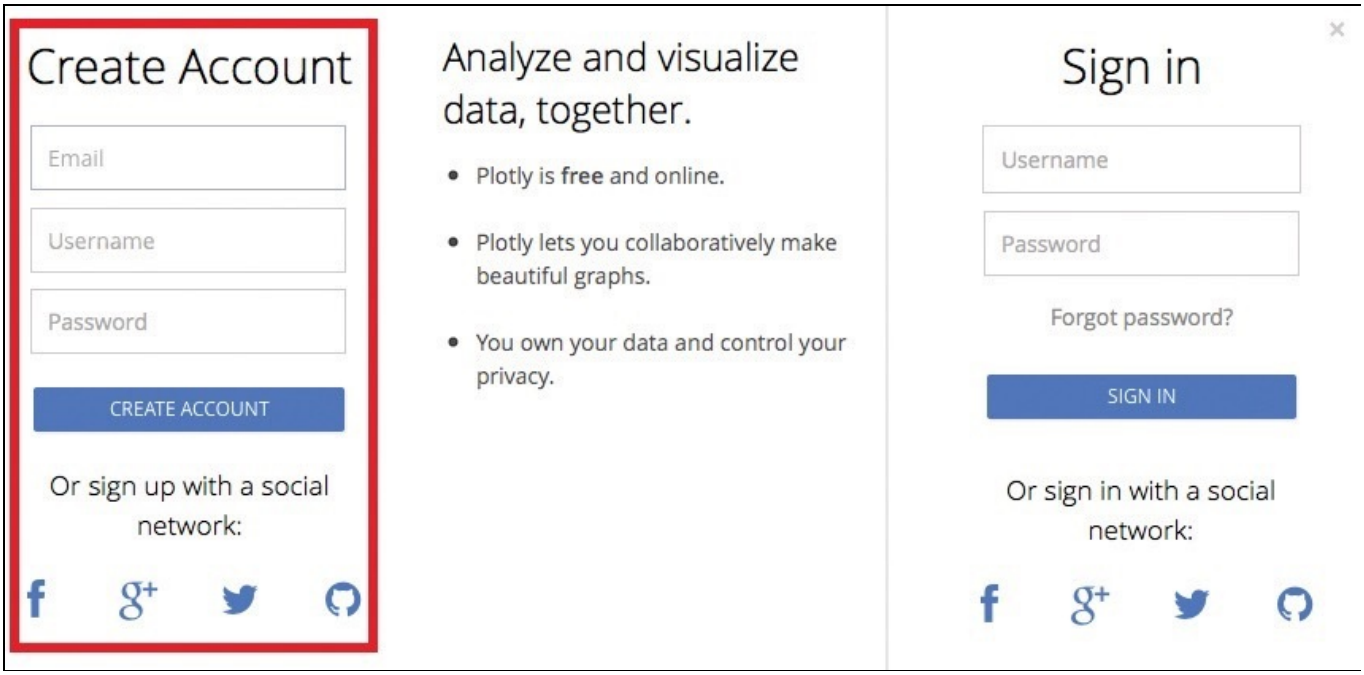


Figura 5.11 Per registrarti su Plot.ly devi scegliere indirizzo e-mail, nome utente e password, oppure autenticarti con uno dei tuoi profili social.

Esplorando il sito web scoprirai un'intera sezione dedicata all'interazione tra Arduino e Plot.ly. Vai all'indirizzo `https://github.com/plotly/arduino-api` e scarica l'archivio ZIP con tutto il materiale necessario per il collegamento al servizio (trovi il link **Download ZIP** sulla destra nella pagina).

UNA LIBRERIA FLESSIBILE

L'archivio ZIP fornito da Plot.ly attraverso GitHub raggruppa molte sottocartelle: ognuna contiene la libreria specifica per il tipo di shield indicato.

A seconda dell'hardware che hai a disposizione, potrai adattare il tuo sketch e fare in modo che comunichi con Internet e Plot.ly attraverso canali differenti: ethernet, Wi-Fi o, come nel caso descritto in questo capitolo, GSM. Sono presenti anche una libreria specifica per la scheda Arduino Yún e le istruzioni (in inglese) per utilizzare tramite la connessione seriale USB un computer con accesso in Rete. Nel momento in cui si scrive la libreria Plot.ly non è ancora aggiornata per essere utilizzata con la versione 1.6.0 dell'IDE. Questo esempio finale fa perciò riferimento alla versione 1.0.5.

Lo sketch completo

Come già fatto per i pacchetti dedicati ai sensori, spostiamo la cartella **plotly_streaming_gsm** in **Documenti/Arduino/libraries** insieme alle altre librerie installate. Riavviando l'IDE avremo a disposizione le nuove funzionalità.

Ora abbiamo tutti gli elementi per creare un unico sketch: la lettura dei sensori (temperatura, umidità e pressione), la connessione in Rete con lo shield GSM e la trasmissione dei dati a Plot.ly: per ogni sezione dello sketch prenderemo spunto dagli esempi specifici.

Lavorando con uno sketch piuttosto complesso, ti renderai conto che è indispensabile scrivere codice ordinato e ben commentato. Cercheremo di suddividerlo in sezioni per analizzarlo in modo approfondito, soffermandoci sulle sezioni aggiuntive rispetto agli sketch di esempio precedenti.

Per prima cosa è necessario importare le librerie che utilizzeremo:

```
#include <DHT.h>
#include <SFE_BMP180.h>
#include <Wire.h>
#include <GSM.h>
#include <plotly_streaming_gsm.h>
```

Poi passiamo alla definizione dei parametri, prima quelli relativi ai sensori, poi quelli relativi alla connessione GSM e infine quelli riguardanti Plot.ly:

```
// definizione pin per il sensore DHTxx
#define DHTPIN 4
// definizione del tipo di sensore DHTxx
#define DHTTYPE DHT22  // scegli tra DHT11 DHT21 DHT22
// definizione della quota delle misurazioni
#define QUOTA 125.0 // quota di Milano, in metri

// Codice PIN della scheda SIM
#define PINNUMBER ""
// informazioni accesso alla rete GPRS
#define GPRS_APN      "APN"       // GPRS APN
#define GPRS_LOGIN    "login"     // GPRS login
#define GPRS_PASSWORD "password"  // GPRS password

// definizione del numero di tracce di plot.ly
#define nTraces 4
// token per plot.ly: uno per ogni sensore
char *tokens[nTraces] = {"nnn", "nnn", "nnn", "nnn"};
```

Ora possiamo inizializzare tutti gli oggetti che utilizzeremo nelle funzioni principali:

```
DHT dht(DHTPIN, DHTTYPE);  // sensore DHTxx
SFE_BMP180 bmp;            // sensore BMPxxx
// inizializzazione GSM
GSMClient client;
GPRS gprs;
GSM gsmAccess;
// inizializzazione di plot.ly
plotly graph("user", "API", tokens, "meteo", nTraces);
```

NOTA *Trovi il tuo* Username *e la tua* API Key *nella sezione* Settings *di Plot.ly all'indirizzo* `https://plot.ly/settings/api`*. Nella stessa sezione del sito puoi generare i token da inserire per identificare le tracce: a ogni codice, univoco, corrisponderanno i dati di un sensore.*

Segue la funzione `setup()`: come sempre è dedicata alla preparazione dell'ambiente di lavoro. Anche in questo caso cercheremo di suddividerla in più parti per comprenderne meglio il funzionamento.

Avviamo ora la comunicazione seriale (utile per il debug dello sketch) e inizializziamo i sensori DHT e BMP:

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

```
  dht.begin();
  if (!bmp.begin()) {
    Serial.println("Errore BMPxxx");
    while(1); // Pausa in caso di errore
  }
```

Attiviamo la connessione GSM prendendo spunto dal codice presente nello sketch di esempio **GsmWebClient** già citato. Il ciclo `while` viene reiterato mentre la variabile `notConnected` mantiene valore `true`. Quando la connessione inizializzata con i parametri definiti nella parte iniziale dello sketch va a buon fine, la condizione del ciclo `if()` si verifica, e la variabile assume valore `false`, permettendo l'uscita dal ciclo e l'esecuzione delle istruzioni successive. In caso di problemi, sul monitor seriale verrà visualizzato il messaggio **Non connesso** una volta per ogni tentativo di connessione:

```
boolean notConnected = true;
// Inizializzazione shield GSM
while(notConnected) {
   if((gsmAccess.begin(PINNUMBER, false)==GSM_READY)
   & (gprs.attachGPRS(GPRS_APN, GPRS_LOGIN,
   GPRS_PASSWORD)==GPRS_READY)) {
   notConnected = false;
  } else {
    Serial.println("GSM non connesso");
    delay(1000);
  }
}
```

Segue la connessione a Plot.ly: facendo riferimento all'oggetto `graph` definiamo il numero di letture da memorizzare (`maxpoints`) e il fuso orario di interesse (`timezone`). Viene eseguito un solo tentativo con `graph.init()` e, in caso di errore, l'esecuzione dello sketch viene interrotta con il ciclo infinito `while(1)`. In caso contrario viene abilitato il flusso di dati con `graph.openStream()`:

```
  graph.maxpoints = 1000;
  graph.timezone = "Europe/Rome";
  boolean success;
  success = graph.init();
  if(!success) {
    Serial.println("Plot.ly non connesso");
    while(1) {}
  }
  graph.openStream();
}
```

Passiamo ora ad analizzare la funzione `loop()`. La prima istruzione `delay()` definisce la frequenza delle letture. In questo sketch di esempio, vista la tipologia di dati che registreremo, decidiamo di impostare una pausa di un minuto tra una lettura e la successiva:

```
void loop() {
  // una lettura ogni 60 secondi
  delay(60000);
```

Dichiariamo poi le variabili che utilizzeremo nella funzione. Ricorda che, dichiarandole all'interno di `loop()`, a ogni nuovo ciclo verranno inizializzate:

```
// variabili per il sensore DHTxx
float DHTt, DHTh;
// variabili per il sensore BMPxxx
char stato;
double BMPt,BMPp,BMPp0;
```

Proseguiamo con la lettura del sensore DHT, prendendo spunto dallo sketch di esempio della relativa libreria. In particolare, nota l'istruzione `return` legata alla verifica dei dati letti: in caso di errore l'esecuzione di `loop()` verrà interrotta e riavviata, per procedere con una nuova lettura. Non sarà possibile proseguire senza dati validi:

```
// lettura temperatura dal sensore DHTxx
DHTt = dht.readTemperature();
// lettura umidità dal sensore DHTxx
DHTh = dht.readHumidity();
// verifica dei dati letti
if (isnan(DHTt) || isnan(DHTh)) {
  Serial.println("Errore lettura DHTxx");
  return;  // riavvia il loop in caso di errore
}
```

In modo analogo, la lettura del sensore barometrico BMP prende spunto dallo sketch di esempio della libreria dedicata, e anche in questo caso un controllo finale sulla validità delle informazioni impedisce l'invio a Plot.ly di dati corrotti:

```
// lettura temperatura e pressione da BMPxxx
stato = bmp.startTemperature();
if (stato != 0) {
  delay(stato);
  stato = bmp.getTemperature(BMPt);
  if (stato != 0) {
    stato = bmp.startPressure(3); // precisione da 0 a 3
    if (stato != 0) {
      delay(stato);
      stato = bmp.getPressure(BMPp,BMPt);
      if (stato != 0) {
        BMPp0 = bmp.sealevel(BMPp,QUOTA);
      }
    }
  }
}
// verifica dei dati letti
if (stato == 0) {
  Serial.println("Errore lettura BMPxxx");
  return; // riavvia il loop in caso di errore
}
```

La sezione terminale di `loop()`, con l'istruzione `graph.plot()`, trasmette a Plot.ly i dati raccolti:

```
  graph.plot(millis(), DHTt, tokens[0]);
  graph.plot(millis(), (float)BMPt, tokens[1]);
  graph.plot(millis(), DHTh, tokens[2]);
  graph.plot(millis(), (float)BMPp, tokens[3]);
  Serial.println("Dati inviati");
}
```

NOTA *Nell'ultimo blocco di istruzioni nota la parola chiave* `(float)` *tra parentesi prima del nome delle variabili* `BMPt` *e* `BMPp`*. Questa sintassi, definita* casting*, permette di convertire tali variabili, inizialmente dichiarate come* `double`*, in* `float`*. Senza questa accortezza il compilatore restituirebbe un errore legato al tipo di dato passato alla funzione* `graph.plot()`*.*

A questo punto lo sketch è davvero completo: controlla i collegamenti dei componenti e fissa lo shield GSM sulla scheda Arduino Uno prima di caricarlo. Per i primi test puoi verificare l'output attraverso la connessione USB e il monitor seriale, ma tieni presente che fornendo alla stazione meteo una sorgente di alimentazione (e un'adeguata protezione dagli agenti atmosferici) potrai posizionarla ovunque, senza essere limitato da connessioni Wi-Fi o altro.

Accedi al tuo account `https://plot.ly` e troverai il grafico **meteo** ad aspettarti nel tuo spazio di lavoro. Facendo clic sul nome del file passerai alla schermata di dettaglio, aggiornata in tempo reale.

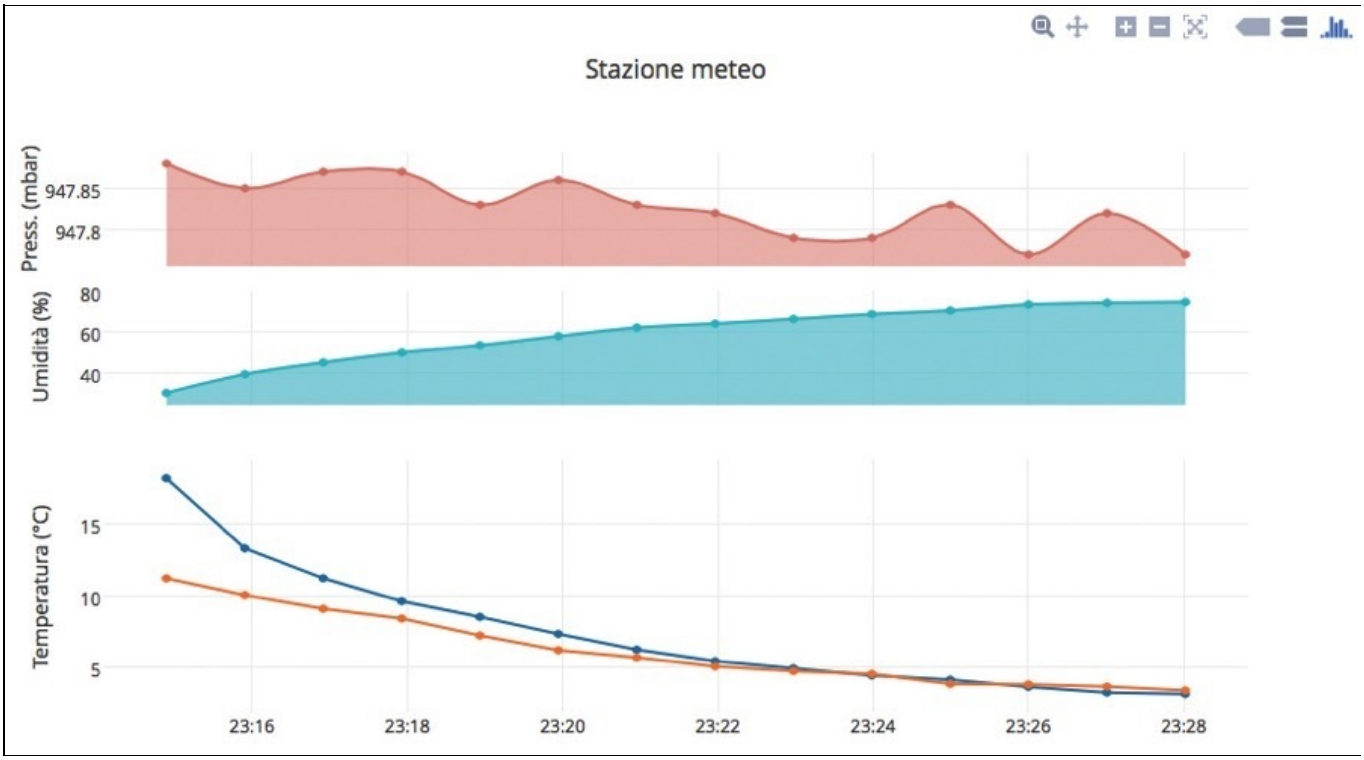


Figura 5.12 Un esempio di grafico realizzato con Plot.ly: intervieni sulle numerose impostazioni e personalizza l'output.

In alto a sinistra trovi gli strumenti che ti permettono di importare dati o visualizzare quelli salvati, salvare ed esportare il risultato e soprattutto personalizzare l'aspetto del grafico (**Traces**, **Layout**, **Axes**, **Notes** e **Legend**). Sperimenta con i vari parametri per ottenere il risultato che desideri.

> **NOTA** *In particolare, con il pulsante* View Data *puoi passare alla visualizzazione della tabella contenente i dati di riferimento dei grafici.*

Con il pulsante **Share**, all'estrema destra, accedi agli strumenti di condivisione della tua creazione: puoi decidere di pubblicare un link sui principali social network e persino generare il codice di *embed*, utile per esempio quando vuoi incorporare il grafico nella pagina di un blog.

Rendi lo sketch ancora più sofisticato

Lo sketch che abbiamo realizzato in questo capitolo è un ottimo punto di partenza per sperimentare con funzionalità ancora più avanzate. Ecco alcuni spunti da cui partire.

Integra altri sensori

Aggiungi sulla breadboard altri sensori e nello sketch le routine necessarie per interpretarne le letture. Potrai arricchire i grafici con altre informazioni: qual è il livello di luce dell'ambiente? Sta piovendo? A che velocità soffia il vento? Puoi davvero dare sfogo alla fantasia: puoi trovare (o costruire) sensori elettronici in grado di comunicare alla scheda Arduino le informazioni più disparate.

Invia SMS dalla stazione

Utilizza le istruzioni `sms.beginSMS()`, `sms.print()` e `sms.endSMS()` come descritto nello sketch di esempio **SendSMS** per inviare messaggi di allerta al superamento di determinate soglie durante la lettura dei sensori: potresti essere avvisato a distanza quando la stazione rileva una temperatura inferiore agli 0 °C, per esempio.

> **NOTA** *Fai attenzione! Nello sketch che abbiamo realizzato i sensori eseguono una lettura al minuto: prevedi una routine in grado di limitare l'invio dei messaggi per riceverli solo quando stabilito e non a ogni lettura che supera la soglia.*

Ricevi SMS sulla stazione

Un altro approccio potrebbe essere quello di collegare alla scheda attuatori in grado di intervenire sull'ambiente in cui si trova la stazione meteorologica e agire quando necessario in base ai dati visualizzati online. Prova a immaginare per esempio di poter comandare l'irrigazione del tuo giardino a tuo piacimento.

> **NOTA** *Tieni presente che a causa delle istruzioni* `delay()` *presenti nello sketch l'esecuzione delle istruzioni legate alla ricezione di un SMS potrebbe non essere immediata. Puoi modificare lo sketch utilizzando la funzione* `millis()` *per rendere la reazione più puntuale.*

Conclusione

In questo capitolo abbiamo affrontato un progetto più complesso, partendo dall'interpretazione dei dati letti da due sensori più sofisticati di quelli analizzati in precedenza, attraverso le librerie dedicate, e arrivando alla pubblicazione online di un grafico con le rilevazioni attraverso la connessione fornita dallo shield GSM.

Nell'ultima parte del capitolo ci siamo poi brevemente soffermati sulla descrizione di possibili spunti, in perfetto spirito Arduino, per migliorare e arricchire ulteriormente le funzionalità dello sketch.

L’hardware Arduino

La piattaforma Arduino nel tempo si è evoluta, e oggi hai a disposizione diverse schede tra le quali scegliere. Tutte condividono la possibilità di lavorare con semplicità, in completa sintonia con l’ambiente di sviluppo Arduino sul tuo computer. Sperimentando e sviluppando prototipi imparerai a capire di volta in volta qual è la scheda più adatta per il progetto che hai in mente.

Premesso che per iniziare a conoscere la piattaforma, la scheda più indovinata è senza dubbio la Arduino Uno, ecco qui di seguito una panoramica sulle caratteristiche e peculiarità di alcune schede in commercio.

Arduino Uno: l’originale

Arduino Uno rappresenta la scheda Arduino per antonomasia. È quella giusta per chi sta muovendo i primi passi per scoprire questo mondo, grazie al giusto equilibrio tra potenzialità, dotazioni e prezzo.

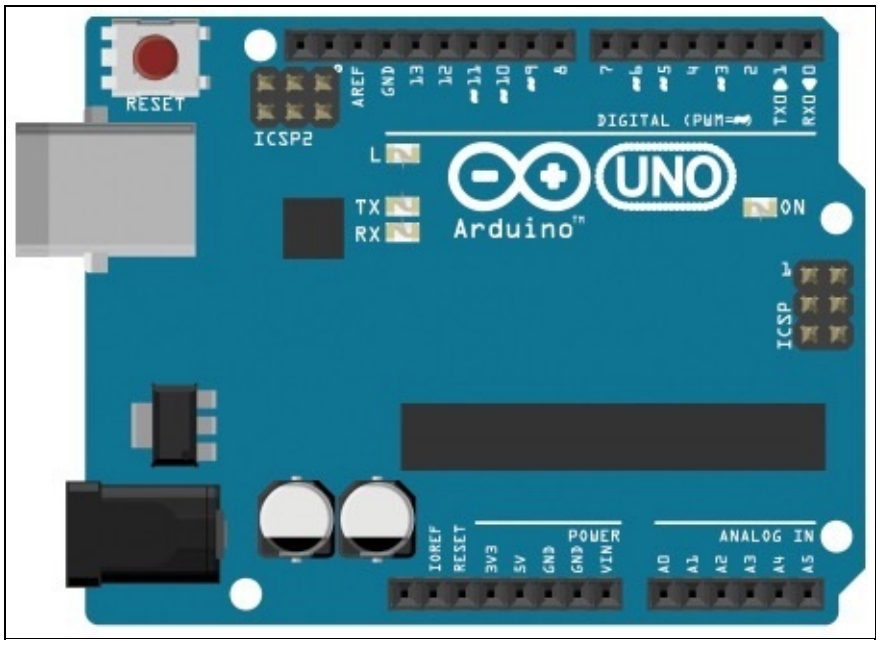


Figura A.1 La scheda Arduino Uno.

Arduino Uno dispone di un connettore USB per interfacciarsi con il computer, di un jack di alimentazione alternativa, di 14 pin di input/output digitali (6 dei quali con funzionalità PWM) e di 6 pin per input analogici.

Integrato sulla scheda, collegato al pin digitale `13`, è presente un LED, molto utile per eseguire test e verifiche senza dover collegare componenti esterni.

La forma della scheda e la disposizione dei pin sono diventati lo standard *de facto* sul quale si basano gli *shield* (schede aggiuntive realizzate per ampliare le funzionalità della scheda base).

Ti renderai conto che questa scheda ha risorse sufficienti per adattarsi a moltissimi progetti, e anzi in molti casi non arriverai neppure a sfruttarne tutte le potenzialità.

UN PO’ DI STORIA

La scheda Arduino Uno attualmente prodotta è il risultato di una continua evoluzione (non ancora conclusa, poiché ogni versione propone aggiustamenti e nuove funzionalità) che ha avuto inizio con i prototipi realizzati a partire dal 2005. Le prime schede disponevano addirittura di una porta seriale, in seguito sostituita dalla più pratica e diffusa connessione USB. Con il tempo anche il microcontrollore su cui si basa Arduino è stato aggiornato con modelli via via più performanti: la potenza di calcolo e le risorse sono di conseguenza aumentate. Da ricordare il precedente modello Arduino Diecimila (che deve il suo nome al numero di schede realizzate nel primo stock di produzione): durante il

trasferimento dello sketch la scheda non doveva più essere resettata manualmente premendo il relativo tasto. Un piccolo, ma fondamentale, passo avanti. Con il passaggio alla scheda Arduino Duemilanove, è stato tra le altre cose eliminato un altro intervento manuale: questa scheda è in grado di identificare autonomamente la sorgente di alimentazione (USB o jack esterno) senza che l'utente la selezioni spostando il relativo jumper, come era necessario fare nei modelli precedenti.

Un sintomo della maturità dell'ecosistema Arduino è anche il nome scelto per l'attuale scheda, Arduino Uno. È un riferimento al rilascio della versione 1.0 dell'IDE con la quale si programma la scheda: le basi del progetto sono ormai solide e sono un ottimo punto di partenza per futuri sviluppi.

Arduino Leonardo

Arduino Leonardo è un’evoluzione diretta di Arduino Uno: le dimensioni restano le stesse, e anche la disposizione dei pin è la medesima. Cambia però il microcontrollore, che diventa un ATmega32u4: questo modello integra al suo interno la comunicazione USB, permettendo dunque l’eliminazione del chip secondario dedicato.

Questo dettaglio fa sì che la scheda possa essere programmata per essere riconosciuta dal computer a cui è collegata come una normale periferica (per esempio un mouse o una tastiera) senza necessità di driver specifici: è ancora più semplice realizzare HID (*Human interface Device*) personalizzati.

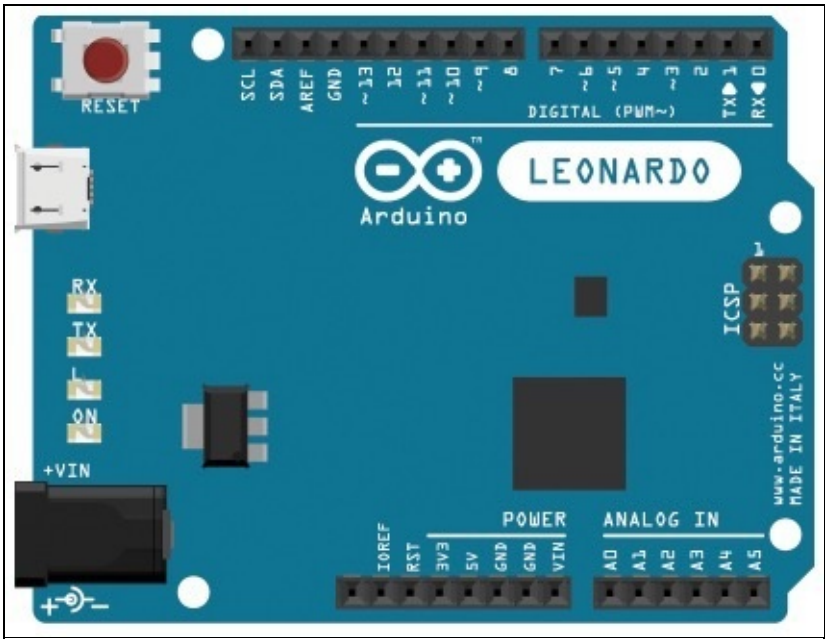


Figura A.2 La scheda Arduino Leonardo.

Tra le altre peculiarità, la scheda Arduino Leonardo mette a disposizione un output PWM anche sul pin `13`, e ben 12 input analogici: oltre a quelli identificati con `A0` … `A5`, anche i pin `4`, `6`, `8`, `9`, `10` e `12` possono essere utilizzati in questa modalità.

Il connettore sulla scheda, per ragioni di diffusione, comodità e ingombro, diventa di tipo micro-USB.

Arduino Micro: le dimensioni contano

Se per un progetto dovessi avere necessità di risparmiare spazio e peso, potresti decidere di utilizzare la scheda Arduino Micro, sviluppata in collaborazione con Adafruit: offre le stesse funzionalità di Arduino Leonardo, in una configurazione davvero compatta (48 mm × 18 mm).

I componenti sono distribuiti su entrambe le facce della scheda, per ridurre le dimensioni al minimo indispensabile. Sulla faccia superiore restano comunque accessibili gli elementi più importanti, il tasto `reset`, il LED `L` collegato al pin `13` (come su Arduino Uno) e i LED `TX`/`RX` per la comunicazione con il computer.

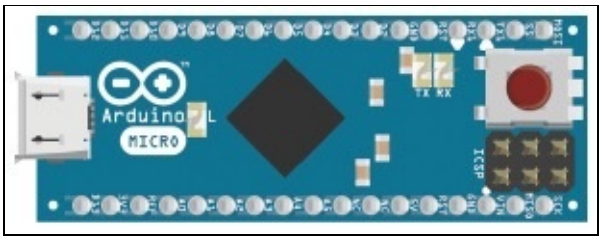


Figura A.3 La scheda Arduino Micro.

Grazie alla spaziatura e alla disposizione dei pin questa scheda può essere inserita in una breadboard, per velocizzare le sperimentazioni senza eseguire saldature. L’alimentazione e la comunicazione con il computer sono garantite anche in questo caso da un connettore micro-USB.

> **NOTA** *In alternativa, quando lo spazio è una priorità, trovi in commercio anche la scheda Arduino Nano, sviluppata da Gravitech, che utilizza lo stesso processore di Arduino Uno.*

Arduino LilyPad: indossa l’elettronica

La famiglia di schede LilyPad è pensata per essere letteralmente cucita sui tessuti: le connessioni tra i componenti possono essere realizzate con filo conduttivo invece che con cavi o piste di rame, permettendoti di entrare nel mondo dei *wearable* con capi di abbigliamento “intelligenti”.

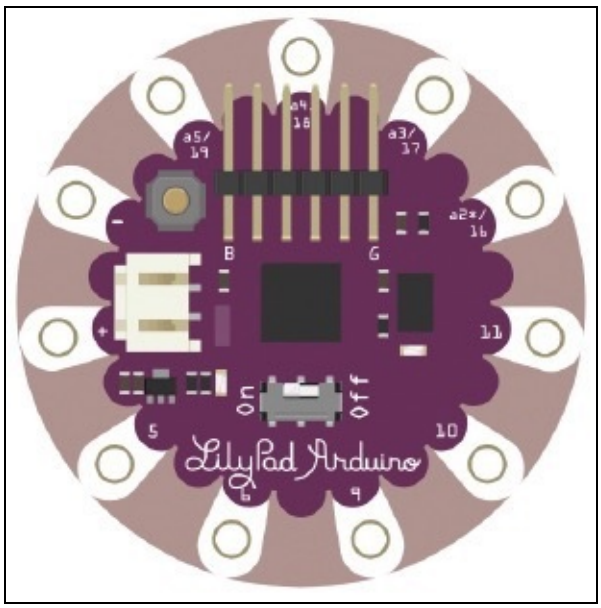


Figura A.4 La scheda LilyPad Arduino Simple.

I pin sono disposti in modo circolare, e la funzione di ciascuno di essi è indicata chiaramente dalla serigrafia sulla scheda.

> **NOTA** *La versione LilyPad Arduino USB, a differenza delle precedenti, integra un connettore micro-USB, un processore ATmega32u4, (lo stesso di Arduino Leonardo) e dispone di nove pin di input/output. La precedente LilyPad Arduino Simple, con connessione seriale, dispone anch’essa di nove pin. La classica LilyPad Arduino, sempre con connessione seriale, dispone di 20 pin di input/output come le normali schede Arduino Uno.*

Oltre alla scheda principale sono stati realizzati vari componenti montati su piccole schede dedicate, pensate apposta per facilitare la connessione e la cucitura sui tessuti. Sul sito SparkFun trovi per esempio sensori di luminosità, pulsanti, LED, moduli di alimentazione e altri componenti specifici che si “abbinano” a questa versione della scheda Arduino.

Il progetto, tuttora in evoluzione, è sviluppato da Leah Buechley e realizzato in collaborazione con SparkFun.

Arduino Mega 2560: molti più input e output

Nonostante esistano sistemi per aumentare la quantità di input e output di un microcontrollore (tecniche di *multiplexing*), potrà capitarti di aver bisogno di un numero di ingressi o uscite maggiore di quello disponibile con Arduino Uno, e di una potenza di calcolo adeguata alla loro gestione: in questi casi Arduino Mega 2560 è ciò che fa per te.

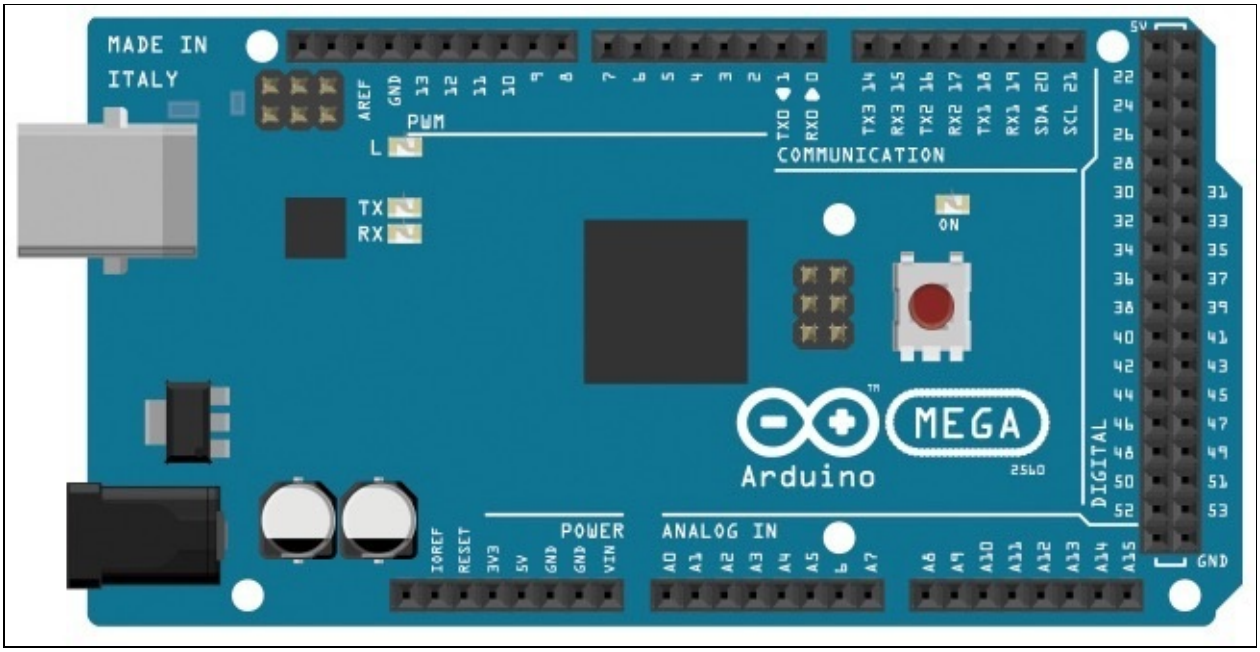


Figura A.5 La scheda Arduino Mega.

Come per Arduino Uno, la connessione USB è utilizzata per la programmazione della scheda, per l'alimentazione e per la comunicazione con il computer. Un jack di alimentazione alternativa è posizionato a fianco del connettore USB, e la sorgente di alimentazione viene selezionata in modo automatico dalla scheda.

Con Arduino Mega 2560 hai a disposizione 54 pin di input/output digitali (14 dei quali con funzioni PWM), 16 pin di input analogico e quattro porte di comunicazione seriali. Anche su questa scheda è previsto il tasto `reset`, a fianco del processore centrale, e per mantenere il più possibile la compatibilità con Arduino Uno al pin digitale `13` è collegato un LED, identificato sulla scheda con l'etichetta `L`.

LA COMPATIBILITÀ CON GLI SHIELD

La disposizione e la funzione dei pin nella parte sinistra della scheda ricalcano quelle standard di Arduino Uno. Questa oculata progettazione permette la compatibilità con la maggior parte degli shield in circolazione: puoi innestarne uno e avere comunque accesso ai pin extra presenti nella parte destra della scheda. Ricorda in ogni caso di verificare sempre la compatibilità degli shield, per evitare problemi durante lo sviluppo e danni all'hardware.

Arduino Due: ancora più potenza di calcolo

La scheda Arduino Due è la prima basata su un processore a 32 bit ARM (rispetto agli 8 bit delle precedenti). La potenza di calcolo, di gran lunga superiore alle schede fin qui descritte, consente applicazioni molto interessanti, come per esempio l'utilizzo del convertitore DAC (*Digital to Analog*) per generare forme d'onda complesse (come un output sonoro).

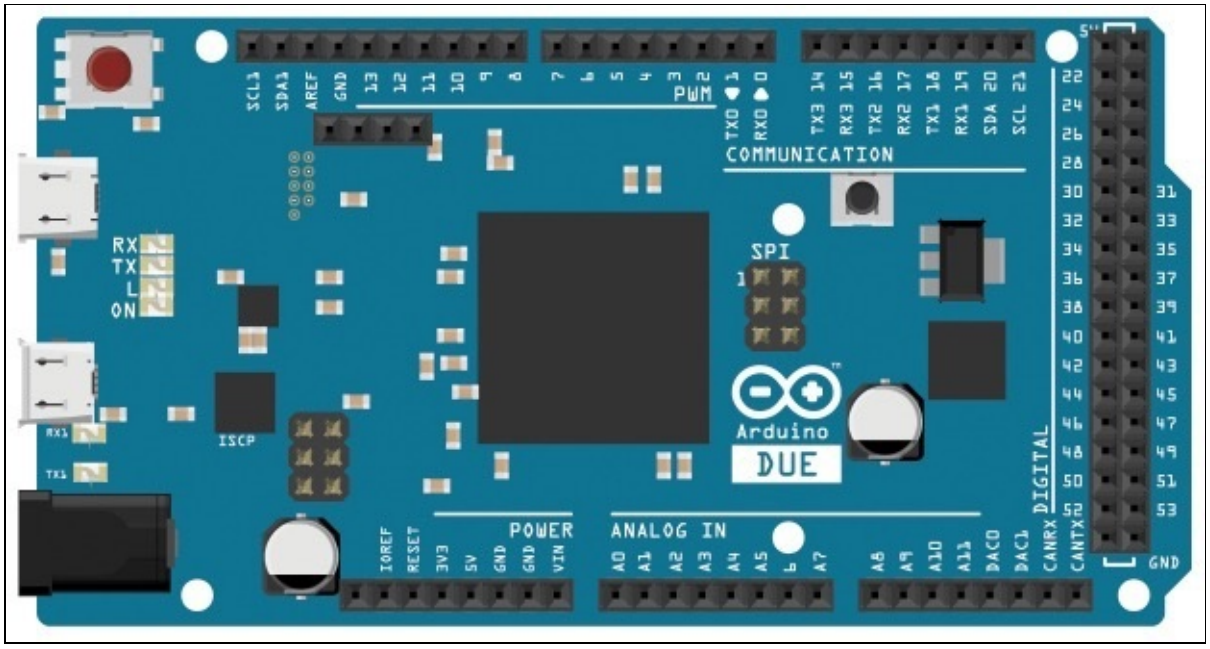


Figura A.6 La scheda Arduino Due.

> **NOTA** *Tieni presente che questa scheda lavora con un voltaggio di 3.3V. Fai attenzione quando colleghi componenti esterni ai pin di input/output, poiché un voltaggio superiore potrebbe danneggiare l'hardware.*

Le dimensioni e la disposizione dei pin, analoghe a quelle di Arduino MEGA, rendono la scheda compatibile con la maggior parte degli shield (se funzionanti a 3.3V), e rende disponibili una porta host USB (per collegare periferiche alla scheda Arduino), 54 pin di input/output digitali (12 dei quali con funzioni PWM), 12 input analogici, 4 porte seriali e il supporto a protocolli di comunicazione più complessi come TWI e CAN.

Intel Galileo: un vero e proprio computer

Ancora non sei soddisfatto delle prestazioni del tuo hardware? Un’occhiata alle caratteristiche della scheda Intel Galileo è sufficiente per intuirne le potenzialità: un processore a 32 bit Intel® Quark SoC X1000 su una scheda completamente compatibile con l’IDE Arduino e con gli shield Arduino Uno.

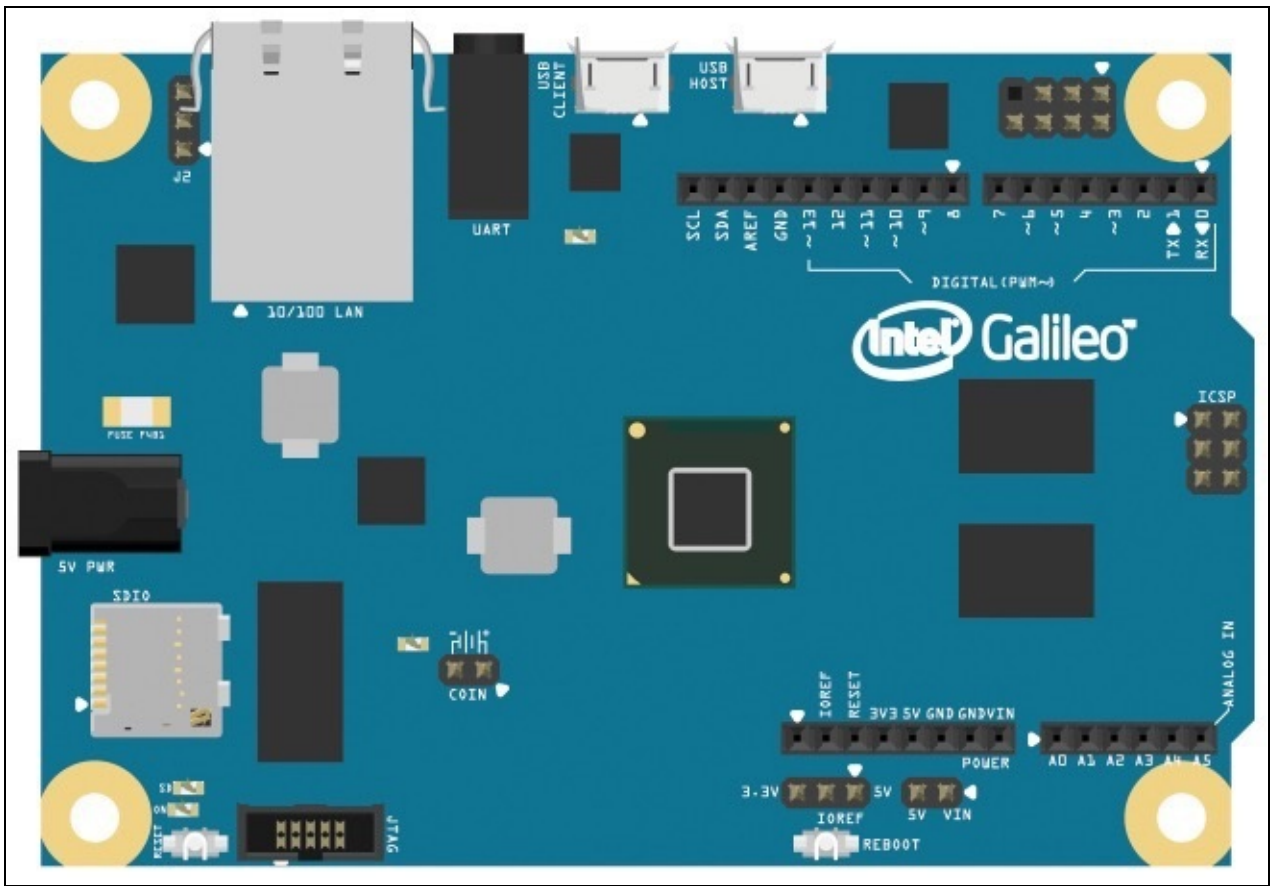


Figura A.7 La scheda Intel Galileo.

Tra le caratteristiche hardware spiccano la porta Ethernet, il lettore di schede MicroSD, la porta host USB e lo slot mini-PCI Express: hai a disposizione moltissimi spunti per scatenare la fantasia e realizzare progetti spettacolari.

NOTA *Un jumper sulla scheda ti permette di selezionare il livello di alimentazione dello shield installato:* `3.3V` *o* `5V`*.*

Arduino Ethernet: per connetterti in Rete

Passiamo ora a un altro ambito: nel **Capitolo 5** abbiamo toccato con mano le possibilità offerte dalla connessione in Rete di una scheda Arduino. Sono senza dubbio interessanti, non credi?

La scheda Arduino Ethernet è una soluzione compatta che permette il collegamento a Internet attraverso una connessione via cavo. Il layout dei pin su di essa è lo stesso di Arduino Uno, ma i pin da `10` a `13` sono riservati alla comunicazione Ethernet: restano a disposizione i pin fino al `9`, quattro dei quali hanno funzionalità PWM.

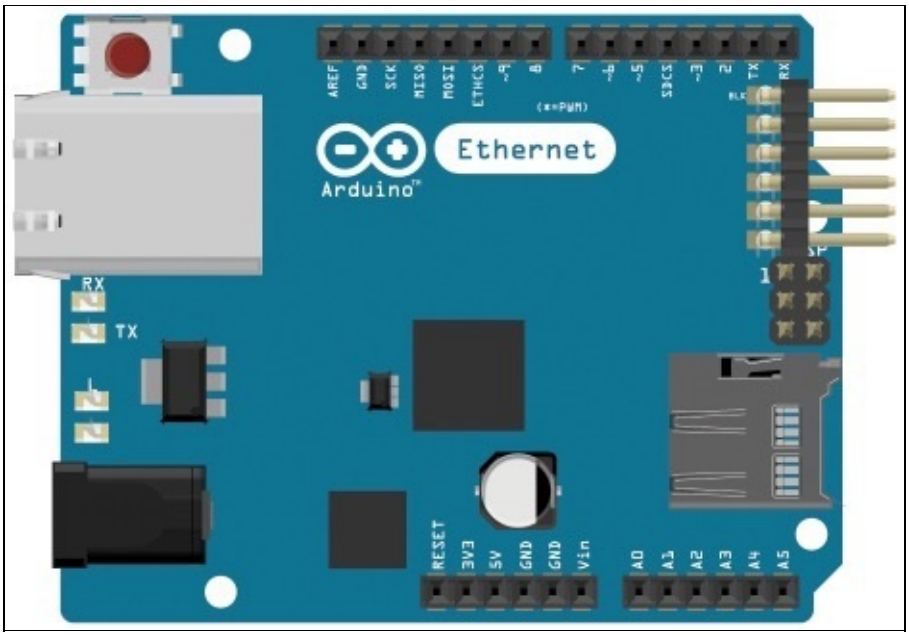


Figura A.8 La scheda Arduino Ethernet.

Questa scheda non dispone di connessione USB, e per trasferire gli sketch è necessario collegare al connettore seriale a sei pin (in alto a destra nella **Figura A.8**) un adattatore USB.

Grazie al lettore di schede MicroSD (in basso a destra) puoi memorizzare dati e avere a disposizione più spazio di archiviazione per l'utilizzo senza computer. Procurandoti un router Ethernet compatibile puoi utilizzare anche la versione di Arduino Ethernet dotata di modulo PoE (*Power over Ethernet*): la scheda verrà alimentata direttamente attraverso il cavo di rete, rendendo superfluo l'alimentatore dedicato e riducendo ulteriormente gli ingombri.

ETHERNET SHIELD

Se preferisci non separarti dalla tua fedele Arduino Uno mentre sperimenti connessioni in Rete, puoi scegliere di innestare su di essa l'Ethernet Shield: avrai a disposizione tutte le funzionalità della scheda Arduino Ethernet, mantenendo la comodità della programmazione tramite interfaccia USB.

Se preferisci, puoi utilizzare il Wi-Fi Shield, analogo a quello appena descritto ma dedicato alle connessioni senza fili, oppure lo shield GSM già utilizzato nel **Capitolo 5**.

Arduino Yún: la scheda per l’Internet delle cose

La scheda Arduino Yún è pensata espressamente per la connessione in Rete: mantenendo le dimensioni di una normale Arduino Uno, ti fornisce l’hardware necessario per connetterti tramite Ethernet o Wi-Fi, e un processore ausiliario dedicato proprio alla gestione del collegamento in Rete.

Il processore principale (lo stesso di Arduino Leonardo) risulta libero dalle pesanti routine riguardanti la connessione in Rete, e il risultato è una scheda semplice da utilizzare, reattiva e snella.

Oltre ai tradizionali LED ON, TX, RX e L, sulla scheda sono presenti altri indicatori dedicati al monitoraggio della connessione in Rete.

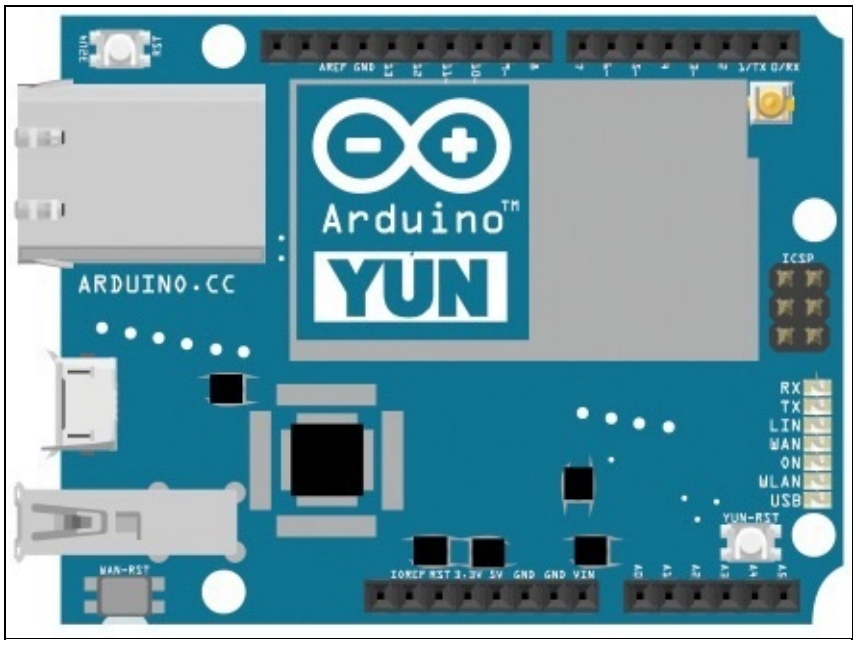


Figura A.9 La scheda Arduino Yún.

È anche importante notare che sono presenti ben tre tasti di reset: 32U4 RST (in alto a sinistra), che riavvia normalmente la scheda, WLAN RST (in basso a sinistra), per ripristinare la connessione Wi-Fi, e YÚN RST (in basso a destra), per riavviare la distribuzione Linux installata sul processore ausiliario.

Per concludere la panoramica delle funzionalità, è utile ricordare la presenza di una porta host USB, di un lettore di schede MicroSD (sulla faccia posteriore) e la possibilità di dotare la scheda di un modulo PoE opzionale.

Strutture di controllo e cicli

Il microcontrollore della scheda Arduino, opportunamente programmato, è in grado di prendere decisioni confrontando valori e verificando condizioni; in un certo senso potremmo dire che le strutture di controllo racchiudono l'intelligenza dei nostri sketch. Di seguito vengono riassunti la sintassi e il funzionamento delle strutture di controllo disponibili nel linguaggio di programmazione Arduino, ereditate da C, e già in parte utilizzate negli sketch di esempio presentati nei vari capitoli.

Prima di iniziare riportiamo per completezza gli operatori che permettono il confronto matematico tra valori:

- `a == b` (`a` è uguale a `b`);
- `a != b` (`a` è diverso da `b`);
- `a > b` (`a` è maggiore di `b`);
- `a < b` (`a` è minore di `b`);
- `a >= b` (`a` è maggiore o uguale a `b`);
- `a <= b` (`a` è minore o uguale a `b`).

Le condizioni

Le strutture più semplici: il codice viene eseguito solo quando una condizione viene soddisfatta.

if … else

L'utilizzo base del costrutto `if` è il seguente:

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

Le istruzioni comprese tra le parentesi graffe `{` e `}` vengono eseguite solo se la condizione dichiarata tra le parentesi tonde `(` e `)` si verifica. In caso contrario l'esecuzione dello sketch prosegue, e il blocco viene ignorato. Puoi omettere le parentesi graffe se l'istruzione da eseguire è una sola, come nell'esempio seguente:

```
if (var > 100) Serial.print("vero");
```

Aggiungendo l'istruzione `else` dopo la chiusura delle parentesi graffe puoi definire un ulteriore blocco di istruzioni da eseguire solo quando la condizione iniziale non si verifica:

```
if (condizione) {
  // istruzioni da eseguire
} else {
  // istruzioni alternative
}
```

Verrà eseguito uno solo dei due blocchi di istruzioni, a seconda dell'esito della verifica sulla condizione iniziale.

È possibile concatenare più verifiche e creare routine di controllo più complesse:

```
if (condizione) {
  // istruzioni da eseguire
} else if (condizione2){
  // istruzioni alternative
} else {
  // istruzioni alternative
}
```

Durante l'esecuzione verrà eseguito solo il primo dei blocchi di istruzioni concatenati la cui condizione sarà verificata, mentre i successivi saranno ignorati: sono blocchi *mutuamente esclusivi*.

NOTA *Anche in questo caso il blocco* `else` *finale non è obbligatorio: omettendolo, se nessuna delle condizioni dichiarate si verifica, non verrà eseguita alcuna istruzione.*

GLI OPERATORI BOOLEANI

È utile ricordare anche gli operatori booleani che consentono di associare più condizioni, per esempio proprio in un costrutto `if()`:

- `&&` rappresenta un *and* logico: entrambe le condizioni devono verificarsi;
- `||` rappresenta un *or* logico: almeno una delle due condizioni deve verificarsi;
- `!` rappresenta un *not* logico: la condizione non deve verificarsi.

Ecco un esempio per capire quando può essere conveniente utilizzare questi operatori: la condizione `if(x > 400 && x < 700)` sarà verificata solo quando la variabile `x` ha un valore compreso tra `400` e `700`).

switch … case

Questo costrutto ha una logica di funzionamento simile a `if … else`, ma la sintassi è differente: per prima cosa viene dichiarata tra parentesi una variabile, poi all'interno delle parentesi graffe viene creato un blocco `case` per ogni valore della variabile da considerare:

```
switch (variabile) {
  case 1:
    // istruzioni da eseguire quando
    // la variabile assume valore 1
    break;
  case 2:
    // istruzioni da eseguire quando
    // la variabile assume valore 2
    break;
  default:
    // istruzioni da eseguire quando
    // la variabile non assume nessuno
    // dei valori previsti
}
```

L'istruzione `break` al termine di ogni alternativa interrompe l'esecuzione delle istruzioni, uscendo dal costrutto `switch` e rendendo anche in questo caso i blocchi mutuamente esclusivi.

L'ultimo blocco, identificato dalla parola chiave `default`, viene eseguito quando nessuna delle condizioni `case` si verifica. È analogo al blocco `else` dopo la condizione `if`. Come `else`, può essere omesso quando non necessario.

Come esercizio, prova a immaginare la stessa struttura descritta nell'esempio sostituendo a `case` una serie di `if` ed `else if`.

I cicli

Spesso si ha l'esigenza di ripetere un blocco di istruzioni più volte. Se ci soffermiamo a ragionare un momento, il blocco `loop()` stesso negli sketch Arduino viene ripetuto a ciclo continuo.

Conoscere le diverse sintassi dei cicli e capire le sottili differenze che li caratterizzano ti permetterà di scegliere sempre quello ottimale per lo scopo che vuoi raggiungere.

for

In questo ciclo la dichiarazione iniziale, tra parentesi tonde, comprende tre elementi separati da punto e virgola: l'inizializzazione di una variabile, la condizione da verificare e l'incremento della variabile.

Tra parentesi graffe sono poi elencate le istruzioni da eseguire:

```
for (inizializzazione; condizione; incremento) {
  // istruzioni da ripetere ciclicamente
  // se la condizione è verificata
}
```

A ogni ripetizione il valore della variabile inizializzata cambierà in base all'incremento dichiarato: se necessario possiamo utilizzarla direttamente all'interno del blocco di istruzioni da ripetere, come in questo esempio:

```
for (int i=0; i<=255; i++) {
  analogWrite(ledPin, i);
  delay(30);
}
```

A ogni ripetizione il valore assunto da `i` sarà diverso, causando un cambiamento nell'effetto dell'istruzione `analogWrite()`.

Quando il valore della variabile non soddisferà più la condizione dichiarata, il ciclo verrà interrotto e lo sketch proseguirà.

> **NOTA** *La dichiarazione iniziale del ciclo* `for`*, ereditata dal linguaggio C, è davvero molto versatile, e ricorrendo ad alcune sottigliezze permette di ottenere comportamenti anche complessi. Nella maggior parte dei casi, comunque, l'utilizzo è quello appena descritto.*

while

Il ciclo `while` richiede nella dichiarazione iniziale solo la condizione da verificare:

```
while (condizione) {
  // istruzioni da ripetere ciclicamente
```

```
  // se la condizione è verificata
}
```

A differenza del ciclo `for` è necessario modificare manualmente all'interno del ciclo il valore della variabile utilizzata per la condizione iniziale. Prendiamo per esempio questo ciclo:

```
int n = 0;
while (n < 10) {
  Serial.println(n);
  n = n+1;
}
```

La variabile `n` ha inizialmente valore `0`, e viene incrementata di `+1` a ogni ciclo. Quando raggiunge valore `10`, la condizione non è più soddisfatta: le istruzioni tra parentesi graffe non vengono più eseguite e lo sketch passa oltre. Considera però quest'altro esempio:

```
int n = 0;
while (n != 1) {
  Serial.println(n);
  n = n+2;
}
```

La variabile `n` parte dal valore `0`, ma viene incrementata di `+2`. Con queste premesse la condizione è sempre soddisfatta, poiché `n` non assumerà mai il valore `1` che renderebbe falsa la condizione interrompendo il ciclo. Presta attenzione durante la definizione delle condizioni: potresti creare cicli infiniti o ottenere comportamenti inaspettati degli sketch.

> **NOTA** *In alcuni casi potrebbe tornarti utile interrompere in maniera indefinita l'esecuzione dello sketch, per esempio in caso di errori: se hai un'esigenza di questo tipo puoi utilizzare la sintassi* `while(true) {}`*, che di fatto crea un ciclo infinito.*

do … while

Questo ciclo è molto simile al ciclo `while` appena descritto. La sola differenza consiste nel fatto che il controllo sulla condizione avviene dopo il blocco di istruzioni e non prima: di conseguenza le istruzioni vengono eseguite almeno una volta in ogni caso:

```
do {
  // istruzioni da ripetere ciclicamente
  // se la condizione è verificata
} while (condizione);
```

Le interruzioni

Possono presentarsi situazioni in cui è necessario scavalcare le condizioni utilizzate per definire i cicli `for`, `while` o `do` e intervenire sulle ripetizioni del ciclo. Ecco le istruzioni che possiamo utilizzare e i dettagli sul loro comportamento.

break

Già accennata nella descrizione del costrutto `switch`, questa istruzione può essere utilizzata anche nei cicli: ne interrompe immediatamente l'esecuzione, passando alle istruzioni successive.

Questa brusca interruzione di un ciclo può tornare utile per reagire prontamente a un input (determinati valori letti da un sensore, per esempio), anche quando siamo nel bel mezzo di un ciclo:

```
while (n < 5000) {
  ledState = !ledState;
  digitalWrite(ledPin, ledState);
  delay(50);
  if (digitalRead(btnPin) == HIGH) break;
  n = n+1;
}
Serial.println("ciclo interrotto");
```

In questo esempio a ogni iterazione di `while` la variabile `ledState` passa da `HIGH` a `LOW` o viceversa. Anche prima della conclusione, se `btnPin` assume valore `HIGH` (ipotizziamo per esempio che venga premuto un pulsante collegato al pin), il ciclo viene subito interrotto e lo sketch passa all'esecuzione delle istruzioni successive.

continue

Utilizzando l'istruzione `continue` all'interno di un ciclo, durante l'esecuzione non la si interrompe del tutto: viene sospesa solo l'iterazione corrente, tornando alla verifica della condizione per dare il via alla ripetizione successiva. Questa istruzione, abbinata a una condizione `if`, permette per esempio di ignorare alcune porzioni dei valori negli intervalli definiti dal ciclo: con una condizione del tipo `if (i > 50 && i < 100) continue;` si ignoreranno i casi in cui `i` ha valore compreso tra `50` e `100`, oppure con `if (i%2 == 1) continue;` verranno saltate le iterazioni in cui `i` assume un valore dispari:

```
for (i=0; i<=255; i++) {
  if(i%2 == 1) continue;
  analogWrite(ledPin, i);
  delay(30);
}
```

Questo ciclo ignorerà le iterazioni in cui `i` assume valori dispari, e di fatto i valori trasmessi a `ledPin` saranno solo `0`, `2`, `4`, `6`, `8` e così via. In alcuni scenari, la possibilità di ignorare certi valori può tornare utile per avere un migliore controllo sul comportamento dello sketch.

goto

Con l'istruzione `goto` puoi modificare il flusso lineare del programma, spostandone l'esecuzione in un punto identificato da un'etichetta.

Ecco un esempio che riprende il funzionamento dell'istruzione `break` descritto in precedenza:

```
inizio:
ledState = !ledState;
digitalWrite(ledPin, ledState;
delay(50);
if (digitalRead(btnPin) == LOW) goto inizio;
Serial.println("ciclo interrotto");
```

La prima riga di questo frammento, `inizio:`, rappresenta la sintassi utilizzata per dichiarare un'etichetta.

Di norma viene sconsigliato l'utilizzo del costrutto `goto` poiché, alterando la linearità dello sketch, è semplice produrre codice poco leggibile e di difficile interpretazione, soprattutto quando la mole di istruzioni è elevata. Sebbene in alcuni casi sia una soluzione comoda, spesso è più sensato ricorrere ad altri stili di programmazione per ottenere gli stessi effetti, evitando complicazioni e producendo codice più leggibile e riutilizzabile.