Corso di programmazione per le scuole con Arduino – PARTE 4

Siamo arrivati all'ultima puntata di questo corso: la volta scorsa abbiamo parlato di come gestire i suoni (tramite microfoni e buzzer). Stavolta presentiamo un singolo progetto che permette di mettere in pratica tutti i concetti di base imparati nelle puntate precedenti. Ma soprattutto, speriamo di stimolare la fantasia degli studenti con un dispositivo che può essere facilmente personalizzato e migliorato. Vedremo, infatti, come realizzare un semplice robot capace di riconoscere le linee disegnate sul pavimento o su un foglio di carta e di muoversi seguendo tali linee. Il dispositivo che progettiamo è molto semplice, ma proprio per questo ogni studente può lavorarci sopra per aggiungere nuove caratteristiche, impiegando in modo creativo le competenze che ha acquisito finora.

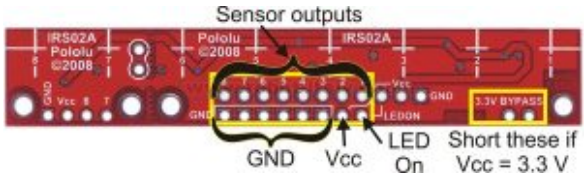
Table of Contents

7 – Un line following robot

Costruire un robot è meno complicato di quanto si possa immaginare, in fondo bastano due servomotori a rotazione continua (continuous rotation servo) e il gioco è fatto. Più

complicato può essere inventarsi un sistema per controllare lo spostamento del robot, ma esiste sempre il meccanismo del "line following", ovvero del seguire le linee. L'idea è semplice: basta disegnare sul pavimento una linea con un buon contrasto (per esempio con un pennarello nero su un foglio di carta bianco). Esiste un metodo piuttosto semplice per permettere ad una scheda Arduino di riconoscere la linea nera: dei sensori infrarossi. Banalmente, basta accoppiare un led a infrarossi con una fotoresistenza (sempre a infrarossi), montandoli sotto al robot. In questo modo, quando la luce infrarossa del led colpisce una zona bianca, la fotoresistenza riceverà un riflesso molto luminoso, mentre quando la luce del led colpisce un punto nero la fotoresistenza riceverà un riflesso molto debole o addirittura nullo (perché il bianco riflette la luce ed il nero la assorbe). Naturalmente il meccanismo funzionerebbe anche con dei normali led colorati, come quelli che abbiamo già utilizzato in precedenza. Però per non avere interferenze dovremmo far correre il robot in una stanza buia, perché la luce del Sole o delle lampade si sovraporrebbe a quella del led. Utilizzando dei led ad infrarossi il problema è risolto, e il vantaggio è che si tratta comunque di banalissimi led, che funzionano come tutti gli altri. Esistono addirittura dei set già pronti con led infrarossi e fotoresistenze, come il QTR-8A, che abbiamo preso come base per il nostro esempio. Il QTR-8A è molto semplice da utilizzare: basta fissarlo sotto al nostro robot, in mezzo ai due servomotori. Il pin Vcc va collegato al pin 5V di Arduino, mentre il pin GND va connesso al GND di Arduino. Poi sono disponibili ben otto pin di segnale: infatti il QTR-8A dispone di 8 coppie di led e fotoresistenze infrarosse, ed ogni fotoresistenza ha un pin che offre a Arduino il proprio segnale analogico, cioè la lettura della luminosità del pavimento. Però Arduino Uno ha soltanto 6 pin analogici: poco male, collegheremo ad Arduino soltanto 6 dei pin del QTR-8A. Per non fare confusione li collegheremo in ordine: il pin 1 del QTR-8A andrà connesso al pin analogico 0 di Arduino, il pin 2 del QTR-8A andrà collegato al pin analogico 1 di

Arduino, e così via. Se avete una scheda più grande, come Arduino Mega, potete utilizzare tutti i pin del QTR-8A, perché un Arduino Mega ha ben 16 pin analogici. In realtà, però, per la maggioranza delle applicazioni 6 coppie di led e fotoresistenze infrarosse sono più che sufficienti.



I contatti della scheda QTR-8A prevedono il Vcc (5V), il GND, e gli input analogici di Arduino

L’idea di base è molto semplice: abbiamo una fila di 6 sensori sotto al robot, ed in teoria vorremmo che la riga nera si trovasse sempre in corrispondenza della metà dei sensori (ovvero tra il terzo ed il quarto). Questo significa che, se facciamo una media della luminosità rilevata dai primi tre sensori ed una media di quella rilevata dagli ultimi tre sensori, è ovvio che dovrebbero essere più o meno uguali. Se la media dei primi tre sensori (che possono essere quelli di sinistra) è più alta significa che la linea nera si trova dalla loro parte, e quindi dovremo far ruotare il robot nella giusta direzione (per esempio destra) per far tornare la linea nera al centro. Infatti, i sensori del QTR-8A funzionano al contrario rispetto ad una normale fotoresistenza: offrono il valore massimo quando la luminosità ricevuta è bassa, e viceversa. Naturalmente, destra e sinistra dipendono dal verso in cui si monta il QTR-8A sul robot: se il robot non si comporta come dovrebbe, basta ruotare di 180° la scheda con tutti i sensori infrarossi.

Utilizzare un radiocomando

Un altro metodo per controllare un robot realizzato con Arduino è utilizzare un radiocomando: i radiocomandi non sono altro che un insieme di potenziometri (le varie leve presenti sul radiocomando sono potenziometri) i cui valori vengono trasmessi a distanza tramite onde radio. Quindi basta collegare il ricevitore del radiocomando ad Arduino, alimentandolo con i pin 5V e GND, e connettendo tutti i vari pin di segnale (ce n'è uno per ciascuna leva del radiocomando) ai pin analogici di Arduino. Arduino può poi leggere i valori dei potenziometri, e dunque capire se sia stata spostata una levetta in tempo reale e reagire di conseguenza (per esempio spegnendo od accendendo uno dei due servomotori).
http://www.instructables.com/id/RC-Control-and-Arduino-A-Complete-Works/?ALLSTEPS

Il codice per il line following

Il codice che fa funzionare questo programma è probabilmente il più complesso che abbiamo analizzato finora, ma è comunque abbastanza semplice da capire:

Prima di tutto si include nel programma la libreria necessaria per il funzionamento dei servomotori.

Dichiariamo poi due variabili speciali, degli "oggetti" (ne avevamo già parlato, sono variabili che possono avere proprietà e funzioni tutte loro), che rappresenteranno i due servomotori. Come abbiamo già accennato, il nostro robot ha un

totale di due ruote motrici, ciascuna mossa da un servomotore: una a destra (right) e l'altra a sinistra (left). Possiamo poi aggiungere una terza ruota centrale, per esempio una rotella delle sedie da ufficio, solo per tenere il robot in equilibrio. Una sorta di triciclo al contrario, visto che le ruote motrici sono due e non una.

Per poter eseguire i nostri calcoli, dovremo tarare i sensori: dovremo capire quale sia il valore massimo (**mx**), minimo (**mn**), e medio (**mid**) di luminosità rilevabile dai vari sensori. Quindi dichiariamo delle variabili che ci serviranno per memorizzare questi valori.

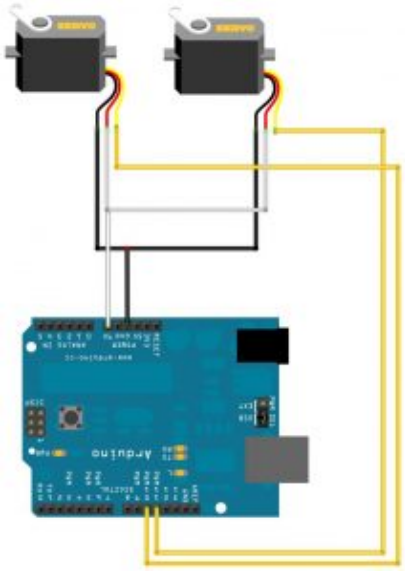
Cominciamo ora a scrivere la funzione **setup**, che viene eseguita all'avvio di Arduino.

Dobbiamo assegnare i due oggetti di tipo servo, **left** e **right**, ai pin digitali di Arduino: abbiamo deciso di collegare il pin **9** al servomotore sinistro ed il pin **10** al servomotore destro. Ricordiamo che devono essere pin PWM, indicati sulla scheda Arduino col simbolo tilde (cioè ~). Indichiamo anche i due valori di minimo e massimo per gli impulsi che faranno muovere il servomotore: di solito non è necessario specificarli (lo fa automaticamente Arduino), ma può essere utile per evitare problemi quando si usano servomotori a rotazione continua come nel nostro caso.

Attiviamo anche la comunicazione sulla porta seriale, così da poter inviare messaggi ad un computer per capire se qualcosa

non stia funzionando nel nostro robot.

Visto che siamo all'inizio, spegniamo il led collegato al pin digitale numero **13** di Arduino (è un led di segnalazione saldato sulla scheda Arduino). Spegniamo anche i due servomotori, così il robot resterà fermo. Con dei servomotori a rotazione continua, lo spegnimento si esegue dando il valore **90** alla funzione write di ciascun oggetto **left** e **right**.



Collegare due servomotori ad Arduino è molto semplice

Calibrare i sensori infrarossi

Attenderemo 5 secondi, ovvero 5000 millisecondi, eseguendo una misura della luminosità di ciascun sensore ogni millisecondo, grazie ad un semplice ciclo **for** che viene ripetuto per 5000 volte.

Accendiamo il led connesso al **pin 13**, quello saldato su Arduino, per avvisare che la calibrazione dei sensori è in atto.

Dobbiamo ora capire quali siano i valori massimo e minimo che si possano misurare con i nostri sensori: per farlo scorriamo tutti i sensori, dal pin analogico **0** al pin analogico **5** di Arduino, leggendo con **analogRead** il valore misurato al momento. Si suppone che il robot si trovi già sopra alla linea, e che almeno alcuni dei sensori siano proprio sopra alla linea nera mentre altri siano sopra al pavimento bianco. Il valore di ogni sensore viene inserito nella variabile **val**. Se tale variabile è maggiore dell'attuale valore massimo (**mx**), allora essa viene considerata il nuovo massimo, assegnando il suo valore a **mx**. Allo stesso modo, se **val** è minore del valore più basso finora registrato **mn**, alla variabile **mn** viene assegnato al valore della variabile **val**.

Prima di terminare il ciclo **for** da 0 a 5000 inseriamo la funzione **delay** chiedendole di attendere 1 millisecondo. Così siamo sicuri che il ciclo durerà in totale almeno 5000 millisecondi, ovvero 5 secondi. In realtà durerà un po' di più, perché le varie operazioni richiedono una certa quantità di tempo, ma sarà probabilmente impercettibile.

Ottenuti i valori massimi e minimi che i sensori possono fornire, possiamo calcolare il valore medio, da memorizzare nella variabile **mid**.

Per segnalare che la calibrazione dei sensori è terminata, spegniamo il led connesso al pin digitale **13** di Arduino.

Finora abbiamo solo calibrato i sensori, ma il robot è ancora fermo. Cominciamo ora la funzione di **loop**, quella che farà muovere il nostro robot.

Per cominciare leggiamo i valori di tutti i sensori, inserendoli in apposite variabili chiamate **s0**, **s1**, **s2**, eccetera.

Iniziamo a muovere il robot: per far girare un servomotore a rotazione continua si può indicare un numero da **0** a **90** oppure da **90** a **180**.
Il valore **180** rappresenta la massima velocità in una direzione, **0** rappresenta la massima velocità nell'altra direzione, e **90** rappresenta la posizione di stallo (quindi il servomotore è fermo).

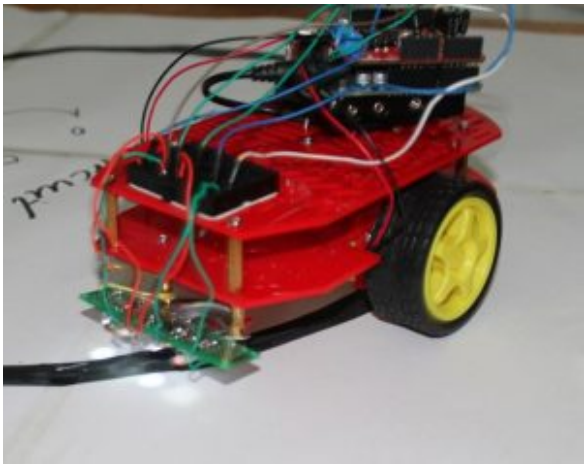
Un semplice miglioramento

Un semplice modo per migliorare il robot può consistere nel rendere variabile la velocità. Se infatti i motori sono sempre impostati a 0 o 180 il robot va sempre alla massima velocità. Si può utilizzare, per esempio, un sensore a ultrasuoni per ridurre la velocità se ci si sta avvicinando a un ostacolo, mappando (funzione **map**) la distanza in un valore da 90 a 0 e da 90 a 180.

Siccome i nostri servomotori sono montati in modo da essere uno speculare all'altro, è ovvio che per far andare il robot avanti uno dei servomotori girerà in una direzione e l'altro nell'altra, così alla fine le due ruote gireranno all'unisono.

Attendiamo un millisecondo, soltanto per essere sicuri che il comando di movimento dei servomotori sia stato applicato.

Abbiamo memorizzato nelle variabili **s0**, **s1**, eccetera, i valori dei vari sensori. Però, come abbiamo detto prima di cominciare a scrivere il programma, noi vogliamo semplicemente comparare la media dei sensori di sinistra con quella dei sensori di destra. Abbiamo deciso che i sensori di **sinistra** siano quelli che sono collegati ai pin analogici **0**, **1**, e **2** di Arduino, mentre quelli di **destra** siano i sensori connessi ai pin analogici **3**, **4**, e **5** (ovviamente dipende da come montiamo il **QTR-8A** sotto al robot). Le due medie si calcolano banalmente con la classica formula matematica: si fa la somma e si divide per 3. Ovviamente, la media potrebbe essere un numero con decimali (con la virgola), ma a noi basta un numero intero: siccome abbiamo definito le due variabili come tipo **int**, Arduino arrotonderà automaticamente i decimali al numero intero più vicino.



Il robot posizionato sopra

la linea nera disegnata su
un pavimento bianco

La linea è a destra o a sinistra?

Normalmente, il robot continua a muoversi in avanti. Però, se la media dei sensori di sinistra è maggiore di quelli dei sensori di destra significa che la linea nera sul pavimento si trova dalla parte sinistra del robot.

Abbiamo indicato anche un fattore correttivo, pari a **240**, per avere un certo lasco: se avessimo scritto soltanto **averageLeft>averageRight** il blocco **if** verrebbe eseguito anche per variazioni minime dei sensori infrarossi tra la parte destra e quella sinistra del robot. Ma vi sarà sempre qualche piccola variazione, anche solo per minime interferenze o oscillazioni nella corrente. Inserendo un fattore correttivo ci assicuriamo che la condizione di **if** venga attivata soltanto se la differenza tra la parte destra e sinistra del robot è notevole.

Ovviamente, se la linea nera è alla sinistra del robot, dovremo ruotare il robot verso sinistra in modo da riportarlo in una posizione in cui la linea nera sia esattamente al centro del robot stesso. E per far ruotare il robot verso sinistra dobbiamo tenere fermo il servomotore di sinistra, dandogli il valore **90**, di modo che faccia da perno, e muovere il servomotore di destra con un valore vicino a **180**. Scegliamo un valore inferiore a 180 perché vogliamo che il servomotore

di destra si muova ma non alla sua massima velocità, così il movimento è più lento e più facile da controllare (se si esagera si rischia di finire al di fuori della linea nera).

Prima di concludere il blocco **if** attendiamo una certa quantità di millisecondi, ottenuta come la metà della differenza delle due medie. L'idea è che in questo modo maggiore è la differenza tra i sensori di destra e quelli di sinistra, maggiore è dunque la dimensione della linea nera, e quindi maggiore è il tempo necessario durante lo spostamento del robot per riuscire ad arrivare ad avere la linea nera al centro del robot stesso.
Siccome la differenza dei due valori potrebbe essere un numero negativo, utilizziamo la funzione **abs** per ottenere il valore assoluto, cioè ottenere la differenza senza segno (in pratica, un eventuale valore **-500** diventerebbe semplicemente **500**).

Se la media dei sensori di sinistra è inferiore a quella dei sensori di destra (tenuto sempre conto del solito fattore correttivo), allora significa che la linea nera è posizionata dalla parte destra del robot. Quindi faremo esattamente l'opposto del precedente **if**: terremo fermo il servomotore destro e muoveremo quello sinistro (anche in questo caso non imposteremo la sua velocità al valore massimo, cioè **0**, ma un po' meno, cioè 40). Anche prima di concludere questo blocco if, attenderemo di nuovo una manciata di millisecondi con lo stesso calcolo precedente.

I percorsi disegnati con nastro adesivo nero su un pavimento chiaro possono anche essere molto lunghi ed elaborati

La fine del percorso

Abbiamo detto al robot cosa fare se la linea nera si trova alla destra oppure alla sinistra del robot stesso. E se invece la linea nera coprisse tutto il pavimento? Significherebbe che il percorso è terminato. Infatti, quando disegniamo il percorso sul pavimento, a meno che non sia un circuito chiuso su se stesso come quelli automobilistici, possiamo indicarne la fine dipingendo di colore nero un bel rettangolo perpendicolare all'ultima parte della linea.

In questo modo quando il robot ci arriva sopra si accorgerà che tutti i suoi sensori, in particolare il sensore **s0** ed il sensore **s5**, che sono il primo e l'ultimo, hanno un valore che è superiore alla media. Questo significa che tutti i sensori sono contemporaneamente sopra alla linea nera, e quindi si è raggiunto il termine del percorso.

In questo caso dobbiamo chiaramente fermare il robot, dando il valore **90** a entrambe i servomotori (come abbiamo già detto,

questo valore provoca l’arresto immediato dei servomotori a rotazione continua).

Per segnalare di avere raggiunto quello che riteniamo essere il termine del percorso (e che quindi il robot non si è fermato per un errore), facciamo lampeggiare rapidamente il led collegato al pin digitale **13** di Arduino, quello saldato sulla scheda. Per farlo lampeggiare 50 volte basta un ciclo **for** che si ripete per l’appunto 50 volte, e ad ogni iterazione non fa altro che accendere il led portando il suo pin al valore **HIGH**, attendere **100** millesimi di secondo, spegnere il led scrivendo il valore **LOW** sul suo pin, ed poi attendere altri **100** millisecondi prima di passare all’iterazione successiva.

Ora attendiamo 5 secondi per essere sicuri che tutte le operazioni necessarie allo spegnimento dei servomotori siano state portate a termine. Durante questi 5 secondi si può tranquillamente spostare il robot, magari posizionandolo di nuovo all’inizio del percorso per farlo ripartire.

La funzione loop si conclude qui, e con essa il programma. Naturalmente, ricordiamo che la funzione loop viene ripetuta di continuo, quindi il robot continuerà a muoversi lungo la linea nera tracciata sul pavimento bianco finché non lo fermiamo facendolo passare sopra ad un rettangolo nero largo tanto quanto l’intera scheda **QTR-8A**, in modo che tutti i sensori infrarossi si trovino contemporaneamente sopra al colore nero.

Costruire un vero Go-Kart con Arduino

Il robot che presentiamo è pensato per essere piccolo e semplice. Ma il bello di Arduino è che si può facilmente salire di scala: basta avere motori più potenti ed un buon telaio, e si può costruire un Go Kart capace di trasportare una persona, controllando il motore elettrico (ne basta uno, visto che la trazione è posteriore) con Arduino. Il progetto che vi suggeriamo utilizza un motore brushless, un Savox BSM5065 450Kv. I motori brushless sono una via di mezzo tra servomotori ed i normali motorini elettrici a spazzole. I motori brushless eseguono rotazioni continue (quindi non si fermano a 180°), e sono controllabili con Arduino (si può controllare la direzione e la velocità). Hanno anche il notevole vantaggio di poter fornire molta potenza e dare dunque una notevole velocità al mezzo su cui vengono montati. Inoltre sono quasi immuni all'usura, rispetto ai motori a spazzole. Ne esistono di piccolissimi, per progetti di modellismo, oppure di enormi (le automobili elettriche usano motori brushless).
http://www.instructables.com/id/Electric-Arduino-Go-kart/?ALLSTEPS

Il codice sorgente

Potete trovare il codice sorgente dei vari programmi presentati in questo corso di introduzione alla programmazione con Arduino nel repository:
https://www.codice-sorgente.it/cgit/arduino-a-scuola.git/tree/
Il file relativo a questo articolo è 7-robot-linefollowing.ino.

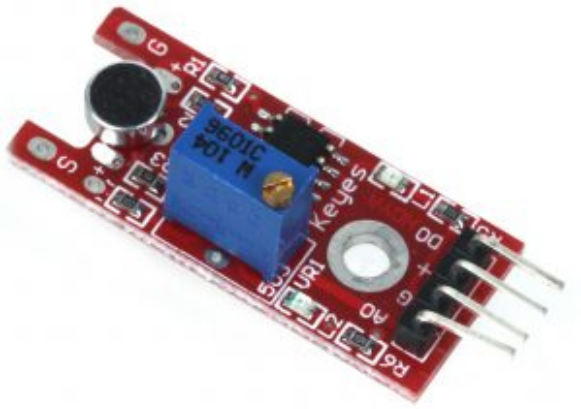
Corso di programmazione per le scuole con Arduino – PARTE 3

Nella scorsa puntata di questo corso abbiamo presentato una serie di esempi per imparare a programmare con Arduino partendo da zero, rivolti soprattutto a studenti, insegnanti, designer, e altre persone che non sono già abituate a programmare. Vogliamo ora proseguire con degli altri esempi, un po' più avanzati ma comunque abbastanza semplici, spiegandoli nei dettagli. Spiegheremo in particolare come utilizzare un microfono per riconoscere suoni come il battito delle mani o un fischio, e un buzzer (i piccoli altoparlanti a cristallo di quarzo) per suonare delle melodie. Ma spiegheremo anche come controllare un relay, grazie al quale diventa possibile utilizzare Arduino per accendere e spegnere a comando elettrodomestici di vario tipo come lampade o stufe elettriche a 220Volt. Le applicazioni di questi esempi sono quindi valide per insegnare ai bambini delle scuole primarie e secondarie di primo grado la logica di base, ma anche per i designer che vogliono realizzare opere d'arte interattive.

5 Accendere un relay con un microfono

Il primo esempio che vediamo è molto semplice ma anche molto elegante ed utile. Proveremo, infatti, ad accendere un relay: i relay (o relé) sono dei semplici interruttori che possono accendere (o spegnere) dispositivi alimentati con un alto voltaggio, come la normale 220V delle prese elettriche di casa controllabili con Arduino. Ogni relay ha due pin da collegare ad Arduino (uno al GND e l'altro ad un pin digitale, come un

led, ed eventualmente anche un ulteriore pin ai 5V di Arduino), e due pin cui collegare il cavo della corrente (per esempio quello di una lampada, al posto di un normale interruttore). Con Arduino possiamo accendere il relay come se fosse un normale led, semplicemente impostando il valore HIGH sul suo pin digitale. Quindi possiamo decidere di accendere il relay in qualsiasi momento: per esempio, possiamo collegare un microfono ad Arduino (noi ci siamo basati sul semplice ed economico KY-038) ed accendere o spegnere il relay quando viene misurato un valore abbastanza forte (per esempio quando qualcuno batte le mani vicino al microfono).



Il microfono KY-038 per Arduino

Il codice del programma da scrivere nell'Arduino IDE e caricare sulla scheda comincia con la classica dichiarazione delle variabili:

Prima di tutto indichiamo i pin che stiamo utilizzando: la variabile **relay** conterrà il numero del pin digitale cui abbiamo collegato il relay, mentre **sensorPin** contiene il numero del pin analogico cui abbiamo collegato il segnale del microfono. I microfoni, infatti, sono sensori analogici.

Nella variabile **sensorValue** inseriremo il valore letto dal sensore, che quindi sarà rappresentato da un numero compreso

tra 0 e 1023 perché questo è l'intervallo dei sensori analogici.

Definiamo poi una variabile di tipo **bool**, ovvero booleano. Una variabile booleana può avere due soli valori: vero o falso, **true** o **false** in inglese. La utilizzeremo per memorizzare l'attuale stato del relay, e infatti la chiamiamo **on**. Se la variabile **on** è **true** vuol dire che il relay è acceso, altrimenti è spento.

La funzione **setup**, eseguita una sola volta all'avvio di Arduino, predispone il pin digitale cui è collegato il relay in modalità di **OUPUT**.
Sfruttando la funzione **digitalWrite** si può quindi scrivere il valore iniziale del relay, che sarà **LOW**, ovvero spento.

Dove trovare i componenti?

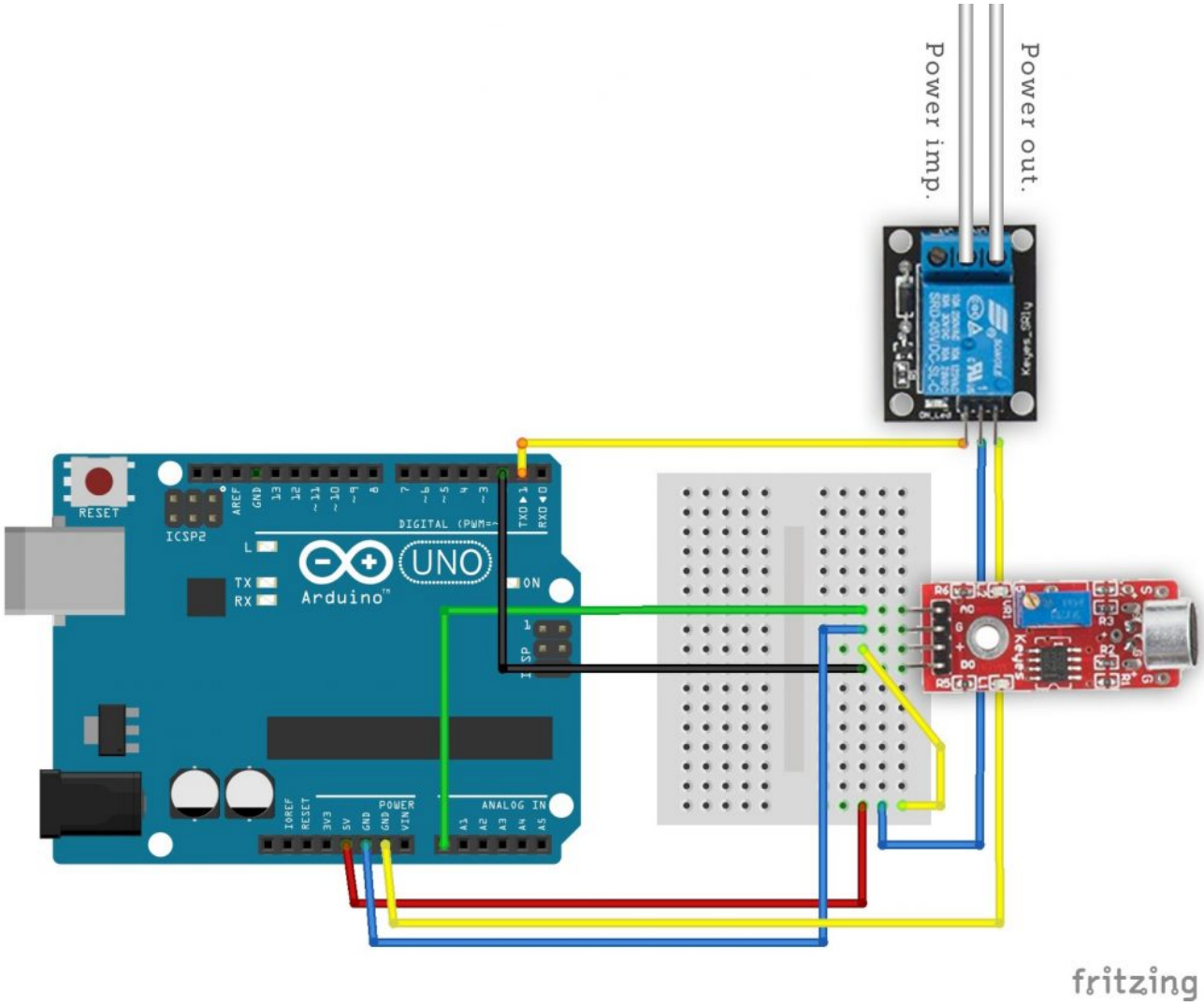
Negli esempi proponiamo dei sensori e degli altri componenti specifici. Mentre un relay si può acquistare in qualsiasi negozio di elettronica, ed anche un pulsante od un buzzer, il microfono e la scheda con i vari sensori infrarossi sono più rari. Ma possiamo trovare tutti questi componenti, eventualmente indicando le sigle che abbiamo suggerito (KY-038), su ebay e su Aliexpress a prezzi molto bassi. Bisogna solo prestare attenzione alle diverse versioni disponibili: spesso, le schede che costano meno richiedono ancora qualche saldatura, mentre ne esistono altre, che costano pochi euro in più, già dotate di connettori di semplice utilizzo.

Poi abilitiamo anche la porta seriale, così sarà possibile scrivere un messaggio al computer eventualmente collegato ad Arduino per fargli sapere cosa stiamo misurando con il

microfono.

La funzione **loop** viene ripetuta all’infinito finché Arduino è acceso, quindi è quella che utilizziamo per realizzare le attività del nostro progetto. Per cominciare, a ogni ciclo provvediamo a leggere l’attuale valore del microfono, che ovviamente è un numero compreso tra 0 e 1023 a seconda del volume percepito dal microfono, memorizzandolo nella variabile **sensorValue**.

Ora possiamo scrivere il numero ottenuto sulla porta seriale, così possiamo controllarlo con un computer. Leggere il valore può essere utile per capire se il microfono debba essere regolato (di solito c’è una apposita rotella) in modo da non ottenere numeri troppi alti o troppo bassi.



Tipico collegamento di microfono e relay ad Arduino

Distinguere un suono dal rumore di fondo

Ora dobbiamo decidere la soglia oltre la quale consideriamo il suono registrato dal microfono adeguato a causare l'accensione o lo spegnimento del relay.

Un semplice **if** ci permette di risolvere il problema, e possiamo scegliere qualsiasi valore come soglia: noi abbiamo scelto 500, ma potete alzarlo o abbassarlo per vedere cosa funziona meglio con il vostro microfono. Se la soglia è stata superata, quindi è stato percepito abbastanza rumore, dobbiamo

agire sul relay. Ma come? Semplice: se il relay è acceso lo vogliamo spegnere, se invece è spento lo vogliamo accendere. In poche parole, dobbiamo invertire il valore della variabile **on**, che indica l'attuale stato di accensione del relay, e che dovrà passare da **false** a **true** e viceversa. Possiamo farlo con l'operatore logico **NOT**, ovvero il punto esclamativo. In poche parole, se **on** è **false**, **!on** sarà **true** e viceversa. Quindi scrivendo **on = !on** abbiamo semplicemente detto ad Arduino di invertire il valore della attuale variabile **on**, scegliendo il suo contrario.

Ora possiamo scrivere l'attuale valore della variabile **on**, sia esso true o false, sul pin digitale del relay. Infatti, in Arduino il valore **true** corrisponde al valore **HIGH**, mentre il valore **false** corrisponde al valore **LOW**. Quindi, avendo una variabile di tipo **bool**, possiamo assegnare direttamente il suo valore ad un pin digitale.

Prima di concludere la funzione **loop**, e il programma, chiamiamo la funzione **delay**, che si occupa solo di attendere un certo numero di millisecondi prima che la funzione **loop** possa essere ripetuta. Abbiamo scelto di attendere 100 millisecondi, vale a dire 0,1 secondi, perché è il tempo minimo per assicurarsi che un rumore rapido come il battito di due mani sia effettivamente terminato, e non venga contato erroneamente due volte. Per essere più sicuri di non commettere errori, possiamo aumentare questo tempo a 1000 millisecondi o anche di più. Il programma è ora completo.
Per spiegare in modo più preciso il funzionamento dell'operatore logico **!**, chiariamo che la riga di codice **on = !on** equivale al seguente **if-else**:

La singola riga di codice che abbiamo utilizzato rende il programma più semplice e più elegante, ma ha esattamente lo stesso significato e lo stesso risultato.

Attenzione alla corrente

Quando lavoriamo con Arduino, stiamo lavorando con l'elettronica, e dunque con della corrente. Ma si tratta di corrente continua a basso voltaggio. Quando aggiungiamo un relay le cose cambiano, perché stiamo andando ad utilizzare anche la normale corrente alternata a 220V delle prese di casa. Ed è molto pericoloso. Le saldature devono essere fatte bene, per evitare possibili cortocircuiti, e non si devono mai toccare contatti scoperti finché la corrente è in circolo. Non dovrebbe mai essere permesso a minorenni di toccare cavi preposti alla conduzione della corrente ad alto voltaggio, anche quando tali cavi siano scollegati dalla presa a muro. Comunque, è bene assicurarsi di lavorare dietro un salvavita, così un eventuale scarica di corrente verrebbe interrotta prima di fulminare un malcapitato. Realizzando questi esempi a scuola conviene sostituire la corrente 220V con un semplice alimentatore da 12V: oggi si trovano molti led che possono essere alimentati direttamente a 12 Volt, riducendo il rischio di farsi male. Il concetto rimane comunque lo stesso, visto che un relay da 220V può tranquillamente essere usato per la corrente 12V continua o alternata.

6 Un allarme sonoro collegato ad una porta

Come si costruisce un sistema di allarme? L'idea è di base è molto semplice, tutto quello che serve è un sensore che rilevi una intromissione, ed un dispositivo sonoro come un altoparlante od un buzzer. Per il nostro esempio sceglieremo un buzzer, anche noto come cicalino o altoparlante piezoelettrico. Ha infatti alcuni vantaggi: è molto economico,

è molto piccolo, e funziona con pochissima corrente quindi non richiede alcuna amplificazione. Per quanto riguarda il sensore, dipende molto da come vogliamo progettare il sistema anti intrusione. L'idea è di controllare una porta: vogliamo un meccanismo che suoni quando una porta viene aperta, e che invece rimanga in silenzio se la porta è chiusa (concettualmente, un po' come la luce del frigorifero, oppure i bigliettini di auguri con la canzoncina).

Un sensore reed può essere recuperato dai contachilometri per biciclette

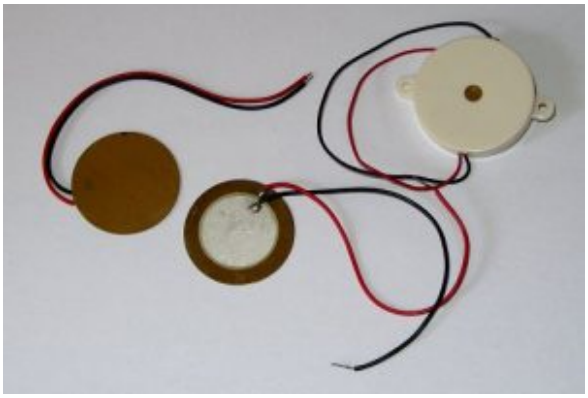
Per questo scopo, in realtà andrebbe bene anche un pulsante: lo si posiziona tra la porta e il muro, così finché la porta è chiusa il pulsante è premuto, mentre appena si apre il pulsante non sarà più premuto. Naturalmente, un pulsante può essere costituito con due pezzi di alluminio (anche quello da cucina in fogli), uno posizionato sulla porta e l'altro sul muro in modo da farli contattare quando la porta è chiusa. In alternativa, si può fissare al muro un sensore reed (anche chiamato reed switch), ed alla porta una calamita: si tratta dello stesso tipo di sensore con cui funzionano i contachilometri per biciclette.
Il codice, che possiamo scrivere direttamente nell'Arduino IDE, è il seguente:

Non servono librerie particolari: tutto ciò che offre il

normale Arduino è più che sufficiente. Però dobbiamo prima di tutto definire le frequenze delle note musicali che ci servono (i valori di riferimento si trovano su https://it.wikipedia.org/wiki/Notazione_dell%27altezza). Per farlo, potremmo dedicare ad ogni nota una variabile, con il valore numerico della frequenza della nota in questione. Ma questo riempirebbe la memoria RAM di Arduino, che è poca. Sarebbe meglio scrivere questi numeri nella memoria flash di Arduino, quella che ospita il codice del programma che carichiamo, perché è molto più grande. Possiamo farlo definendo non una variabile ma una costante, una **costante globale** a essere precisi, con il comando **#define**. In questo modo, per esempio, stabiliamo che la costante **Do4** ha il valore 261, perché la nota do della quarta ottava ha una frequenza di **261 Hertz**.

Definiamo anche una particolare nota con frequenza pari a zero, che utilizzeremo per costituire le pause della musica.

Ora il programma può procedere come al solito, dichiarando due variabili che rappresentino i pin cui sono collegati i componenti elettronici. In particolare, abbiamo deciso di collegare il pulsante (o sensore **reed**) al **pin digitale 2**, ed il **buzzer** al **pin digitale 9**. Il buzzer deve essere collegato ad un pin digitale PWM, indicato dal simbolo ~ sulla scheda Arduino.



Tre buzzer (o cicalini): si

può notare quanto piccoli siano

La melodia da suonare

Ora dobbiamo scrivere le note della musica che vogliamo suonare: le note sono rappresentata da due valori, altezza e durata.

Quindi realizzeremo due diversi **array** (o vettori): un array è, semplicemente una variabile speciale che può contenere molti valori. Praticamente, una lista di valori, ciascuno dei quali potrà poi essere identificato con un numero progressivo tra parentesi quadre, partendo dallo zero. Per esempio, l'elemento **melody[0]** è **La4**, e anche l'elemento **melody[1]** è **La4**, ma l'elemento **melody[2]** è **Si4**.
È possibile costruire array a due dimensioni, praticamente delle tabelle con molte colonne, ma occupano molta memoria e sono scomodi. Meglio dedicare due array diversi alle due diverse informazioni: semplicemente creiamo una lista per l'**altezza** delle note, chiamata **melody**, ed una lista della **durata** delle note chiamata **beats**. Per semplicità, decidiamo che la durata delle note viene indicata come la frazione della nota semibreve. Quindi, se scriviamo 4 intendiamo dire un quarto (ovvero una semiminima), se scriviamo 8 intendiamo un ottavo (una croma), e così via. Naturalmente, si possono anche indicare valori come un terzo o un quinto: non è certo obbligatorio utilizzare numeri pari.

Le note musicali

Arduino è in grado di produrre, con un buzzer tutte le

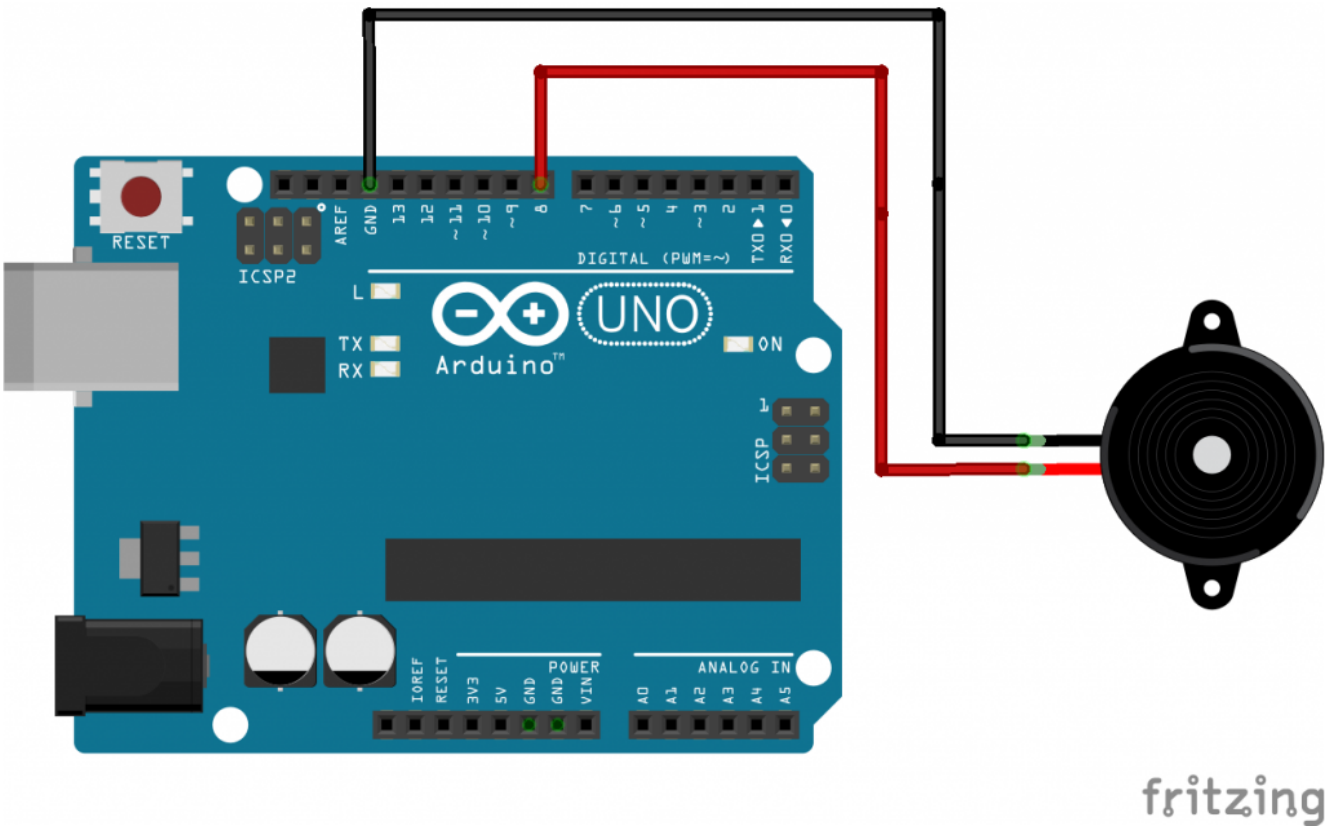
frequenze audio udibili da un essere umano, ed anche alcuni ultrasuoni udibili soltanto da altri animali (come i cani). Le note musicali non sono altro che specifiche frequenze. Le note sono in totale 12, considerando anche i bemolle ed i diesis, e vengono divise in 8 o 9 ottave. Un pianoforte, comunque, non supera l'ottava numero 8. La nota "standard" è il La della quarta ottava, chiamato anche La4, fissato a 440Hz. Il La3 ha una frequenza di 220Hz, mentre il La5 ha una frequenza di 880Hz, e così via.
https://it.wikipedia.org/wiki/Notazione_dell%27altezza

Certo, l'inno alla gioia non è proprio la musica migliore per un allarme. Ma almeno è facilmente riconoscibile, e del resto stiamo realizzando questo esempio per i bambini. Naturalmente, si può scrivere qualsiasi spartito musicale, basta conoscere le note e la loro durata.

Per leggere tutta la melodia dovremo scorrere i due array. E per farlo abbiamo bisogno di sapere quanti elementi sono contenuti nell'array **melody**. Non esiste un metodo diretto per sapere quanti elementi sono presenti dentro un array, però si può utilizzare un trucco: la funzione **sizeof** ci dice la dimensione in byte dell'array. Siccome il nostro array è di tipo **int**, ovvero ogni suo elemento è un numero intero, e in Arduino Uno (con processore a 16 bit) i numeri interi vengono memorizzati in **2 byte**, è ovvio che dividendo la dimensione dell'array per 2 otteniamo il numero di elementi dell'array. Arduino Due ha un processore a 32 bit, quindi per memorizzare i numeri interi utilizza 4 byte invece di due (ogni byte è composto da 8 bit). In quel caso basta dividere per 4 invece che per 2.

Dichiariamo ora due variabili importanti per stabilire la durata generale delle note. La variabile **tempo** contiene, in millisecondi, la durata di una nota semibreve: l'abbiamo impostata a 4000 perché di solito la semibreve viene suonata

in 4 secondi, ma se volete potete modificare l'impostazione per rendere il tutto più veloce o più lento. La variabile **pause**, invece, contiene il tempo che intercorre tra una nota e l'altra: i computer come Arduino sono capaci di suonare le note una dopo l'altra senza alcuna pausa, ma in questo modo si ottiene un suono poco naturale.



Un buzzer può essere collegato ad Arduino connettendo uno dei pin al GND e l'altro ad un pin digitale

Quando a suonare è un essere umano, infatti, c'è sempre una piccolissima pausa tra una nota e l'altra (per esempio il tempo necessario a spostare le dita). Però si tratta di un tempo molto piccolo, quindi lo esprimiamo non in millisecondi, ma in **microsecondi**: 1000 microsecondi sono un millesimo di secondo. È un tempo apparentemente insignificante, ma noterete che senza di esso diventa difficile distinguere il suono delle varie note della melodia.

Ci servono poi tre variabili, che utilizzeremo per capire

quale sia la nota da suonare volta per volta: potremmo dichiararle già nella funzione **loop**, ma lo facciamo nella parte principale del programma così potranno eventualmente essere utilizzate anche in altre funzioni, se qualcuno vorrà migliorare il programma. La variabile **tone_** conterrà la frequenza da suonare, mentre **beat** conterrà la durata della nota espressa come frazione della semibreve. Però Arduino non sa cosa sia la durata di una nota, quindi dobbiamo tradurre la durata delle varie note in millisecondi, ed è quello che faremo con la variabile **duration**. Una nota: la variabile **tone_** non è stata chiamata soltanto tone perché "tone" è anche il nome di una funzione, ed i nomi delle variabili che creiamo devono essere diversi da quelli delle funzioni già fornite da Arduino.

La funzione **setup**, eseguita una sola volta all'accensione di Arduino, non fa altro che attivare i due pin digitali: quello del pulsante sarà un pin di tipo **INPUT**, mentre quello del buzzer sarà ovviamente di tipo **OUTPUT**.

Come funziona un pulsante

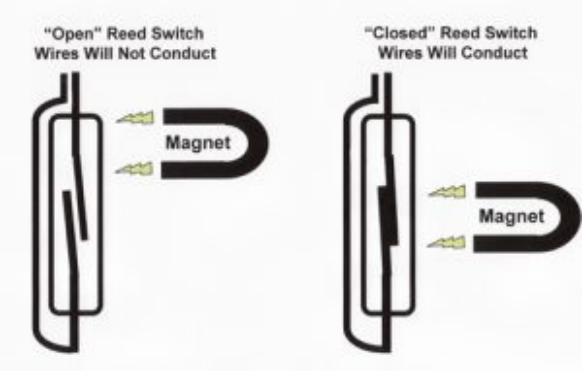
Un pulsante non è altro che un contatto di qualche tipo che ha due posizioni: aperto o chiuso. Quando il contatto è aperto non c'è passaggio di corrente, quando il contatto è chiuso la corrente passa. Un pulsante si costruisce facilmente con Arduino: basta inserire una resistenza da 10KOhm nel pin 5V, ed un cavetto nel pin GND. Il capo libero del cavo va poi collegato al capo libero della resistenza, ed a questa unione va aggiunto un ulteriore cavetto, al quale rimane un capo libero. Quest'ultimo capo libero è uno dei due contatti del pulsante. L'altro contatto si ottiene semplicemente inserendo un cavo in uno dei pin digitali (o analogici, se si vuole) di Arduino, mantenendo libero l'altro capo di questo cavo.

Abbiamo quindi due cavi con un capo libero ciascuno: quando questi si toccano, il pulsante è chiuso, quando non si toccano è aperto. Poi possiamo collegare ai due cavi qualsiasi cosa: un pulsante, un interruttore, un reed, o semplicemente due pezzi di alluminio facendo in modo che volte si tocchino ed a volte no (per esempio fissandoli sul lato di una porta e sul muro).
https://www.arduino.cc/en/Tutorial/Button

Solo se il pulsante non è premuto

La funzione loop è quella che viene eseguita di continuo, quindi è qui che scriveremo il vero e proprio codice "operativo" del programma.

Prima di tutto, dobbiamo occuparci del pulsante: non dimentichiamo che l'idea è di far suonare la melodia solo se il pulsante non è premuto, perché finché è premuto significa che la porta è chiusa. Praticamente, il contrario di un normale pulsante (è come un citofono che suona quando non è premuto invece di suona alla pressione del pulsante).



Un sensore reed è di fatto un pulsante attivato da una calamita

I pulsanti sono fondamentalmente dei sensori digitali, ed offrono ad Arduino due possibili valori: **LOW** (cioè 0 Volt), se il pulsante non è premuto, e **HIGH** (cioè 5 Volt) se il pulsante è premuto. Siccome vogliamo che tutto ciò che segue d'ora in poi avvenga solo se il pulsante non è premuto, controlliamo lo stato del pulsante con la funzione **digitalRead** e verifichiamo grazie a un **if** che tale stato sia uguale a **LOW**.

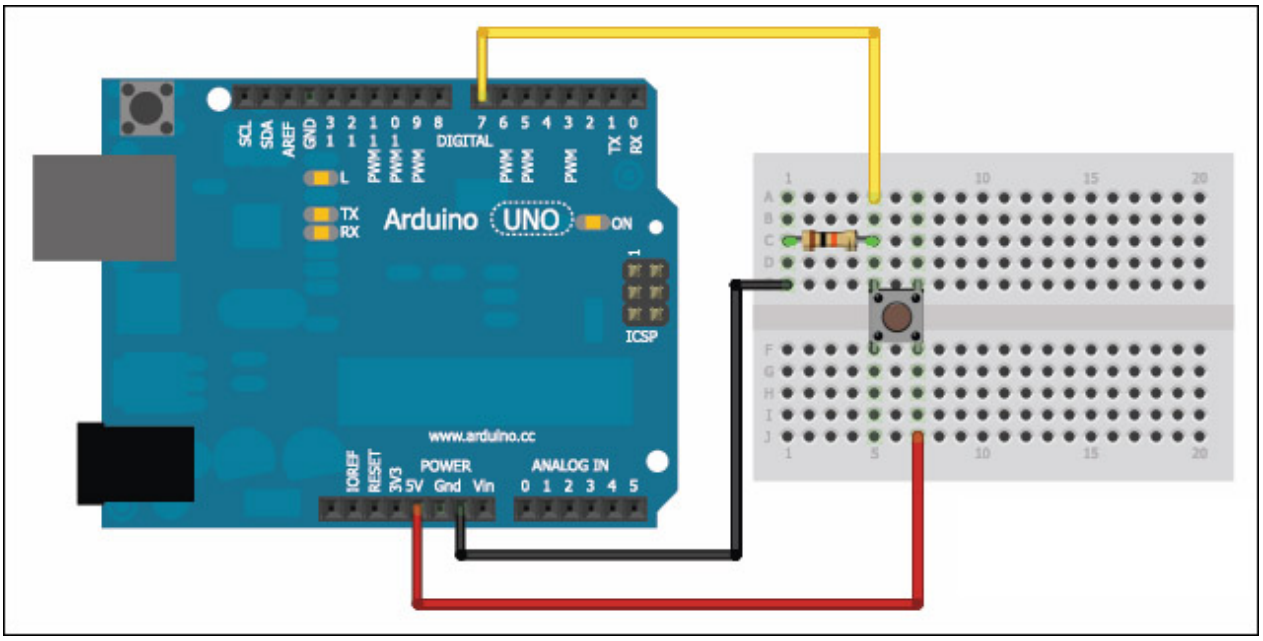
Se il pulsante non è premuto, dobbiamo cominciare a scorrere i due array per leggere le varie note e suonarle. Possiamo farlo con un ciclo **for**: i cicli for hanno una variabile contatore: nel nostro caso la variabile **i**. Il valore iniziale della variabile **i** è **0**, e ad ogni ciclo il suo valore verrà incrementato di **1**, perché questo è previsto dall'ultimo argomento del ciclo for (**i++** infatti significa che ad i va sommato il valore 1). Il ciclo andrà avanti finché la variabile **i** avrà un valore inferiore alla variabile **MAX_COUNT**, che avevamo utilizzato per calcolare il totale degli elementi dell'array. In altre parole, il ciclo comincia con **i** uguale a **0** e termina quando sono stati letti tutti gli elementi dell'array.

Visto che la variabile **i** scorre tutti i numeri dallo **0** al totale degli elementi degli array, possiamo proprio utilizzarla per leggere i vari elementi uno dopo l'altro. Infatti, **melody[i]** è l'i-esimo elemento dell'array che contiene le frequenze delle note, mentre **beats[i]** è l'i-esimo elemento dell'array che contiene le durate delle note. Inseriamo questi valori nelle due variabili che avevamo dichiarato poco fa appositamente. Calcoliamo anche la durata in secondi della nota attuale, inserendo il valore calcolato nella variabile **duration**.

Ora ci sono due opzioni: la frequenza della nota può essere zero o maggiore di zero. Una frequenza pari a zero (o comunque non maggiore di zero) indica una pausa, quindi non si suona

niente, basta aspettare. Un semplice **if** ci permette di capire se la frequenza della nota attuale sia maggiore di zero e quindi si possa suonare.

In caso positivo eseguiamo comunque un controllo: se, infatti, nel bel mezzo della riproduzione il pulsante viene premuto di nuovo (quindi il suo valore diventa HIGH), dobbiamo interrompere la riproduzione della melodia. E per interromperla basta terminare improvvisamente il ciclo for che sta scorrendo tutte le note della melodia.



Il tipico collegamento di un pulsante ad Arduino prevede l'uso di una resistenza da 10KOhm

Questa interruzione può essere fatta con il comando break, che blocca il ciclo for ed esce da esso, facendo sì che il programma termini anche la funzione loop e provveda a ripeterla da capo. Se non si vuole interrompere la riproduzione del suono dopo che essa è già iniziata una volta, basta rimuovere questa riga di codice.

È arrivato, finalmente, il momento di suonare la nota attuale: prima di tutto ci si assicura che il buzzer non stia suonando altre note, per evitare sovrapposizioni. E lo si può fare

chiamando la funzione noTone, indicando il pin digitale cui è collegato il buzzer (nel nostro caso speakerOut). Poi possiamo suonare la nota chiamando la funzione tone: questa richiede tra gli argomenti il pin digitale del buzzer, la frequenza (che è contenuta nella variabile tone_), e la durata della nota. Una cosa interessante della funzione tone è che non blocca il programma fino al termine: vale a dire che anche se noi chiediamo di eseguire una nota di 2 secondi, Arduino non aspetta che il suono della nota sia terminato per procedere con la riga di codice successiva. Questo è molto utile nel caso si abbiano più buzzer collegati ad Arduino e si vogliano suonare contemporaneamente. Però, nel nostro caso, dobbiamo dire ad Arduino di aspettare che la nota sia terminata prima di procedere. Quindi utilizziamo la funzione delay per chiedere ad Arduino di attendere un numero di millisecondi pari proprio alla durata prevista della nota musicale. Inoltre, attendiamo anche un numero di microsecondi pari alla pausa tra una nota e l'altra che avevamo deciso, con la funzione delayMicroseconds.

Come avevamo detto, è possibile che la nota da "suonare" abbia frequenza pari a zero: ed in questo caso è una pausa, quindi non si suona nulla, basta attendere il tempo necessario con la funzione delay. Siccome prima avevamo cercato di distinguere le note vere dalle pause utilizzando un ciclo if, ora basta aggiungere un else per dire ad Arduino cosa fare quando trova una pausa. Terminato anche l'else, possiamo chiudere anche il ciclo for, il ciclo if iniziale (quello che verificava che il pulsante non fosse premuto), e la funzione loop. Il programma è terminato.

Il codice completo

Potete trovare il codice sorgente dei vari programmi

presentati in questo corso di introduzione alla programmazione con Arduino nel repository:
https://www.codice-sorgente.it/cgit/arduino-a-scuola.git/tree/
I file relativi a questa lezione sono, ovviamente, **microfono-relay.ino** e **allarme.ino**.

Corso di programmazione per le scuole con Arduino – PARTE 2

Nella scorsa puntata di questo corso abbiamo accennato ai concetti fondamentali della programmazione, dalle variabili alle funzioni per leggere il valore dei sensori. In questa seconda lezione approfondiamo le funzioni e gli oggetti più utili nel mondo di Arduino. Vedremo come utilizzare i sensori digitali e come accedere a pagine web per estrarre informazioni. È il concetto di API web, molto più semplice nella pratica di quanto la teoria possa far supporre, una abilità fondamentale per realizzare oggetti "intelligenti", in grado di reagire a informazioni che ricevono dall'esterno. Vedremo anche che il controllo di un servomotore non è troppo diverso da quello di un led, quindi gli studenti potranno pensare alla realizzazione di sistemi in movimento.

Naturalmente, qui presentiamo le principali caratteristiche di Arduino con degli esempi pratici, adattabili a vari ambiti, dalla scuola all'arte e il design. Gli insegnanti devono ricordare che si tratta più che altro di spunti utili soprattutto per capire la logica di Arduino e la

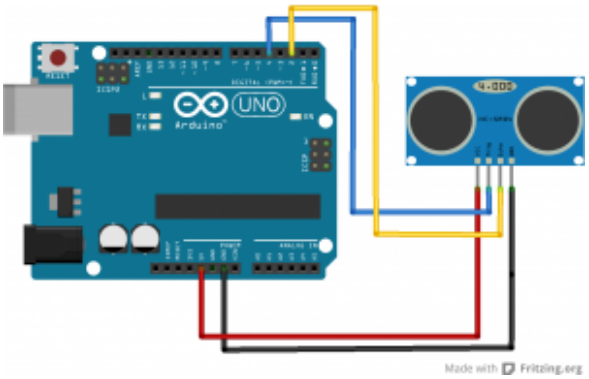
programmazione. Naturalmente ciascuno dei nostri progetti può essere modificato, per esempio usando sensori diversi. Per imparare davvero come funzioni Arduino, infatti, la cosa migliore consiste proprio nel fare molti tentativi, in modo da prenderci la mano e soprattutto sviluppare la fantasia necessaria a risolvere creativamente i problemi che possono presentarsi.

3 Accendere un led in dissolvenza man mano che ci si avvicina ad Arduino

Una cosa interessante dei led è che possono essere accesi "a dissolvenza". Cioè, invece di passare in un attimo dal completamente spento al completamente acceso, possono aumentare gradualmente luminosità fino ad accendersi completamente (e anche viceversa, possono diminuire luminosità gradualmente fino a spegnersi). Un tale comportamento è ovviamente **analogico**, non digitale, perché il digitale contempla solo lo stato acceso e quello spento, non lo stato acceso al 30% oppure acceso all'80%. Arduino non ha pin analogici di output, li ha solo di input, ma ha alcuni pin digitali che possono simulare un output analogico. Questi si chiamano **PWM** (Pulse With Modulation), e sono contraddistinti sulla scheda Arduino tra i vari pin digitali dal simbolo tilde (cioè ~). Se vogliamo poter accendere in dissolvenza un led, dobbiamo collegarlo ad uno dei pin digitali indicati dal simbolo ~, e poi assegnare a tale pin un valore compreso tra 0 e 255 utilizzando non la funzione digitalWrite, che abbiamo visto nell'esempio della puntata precedente, ma la funzione **analogWrite**. Naturalmente, per aggiungere una certa interattività, ci serve un sensore: possiamo utilizzare il sensore **HC RS04**. Si tratta di un piccolo sensore ad ultrasuoni

capace di leggere la distanza tra il sensore stesso e l'oggetto che ha di fronte (rileva oggetti tra i 2cm ed i 400cm di distanza da esso, con una precisione massima di 3mm). Il suo pin **Vcc** (primo da sinistra) va collegato al 5V di Arduino, mentre il pin **GND** (primo da destra) va collegato al GND di Arduino. Il pin **Trig** (secondo da sinistra) va collegato al pin digitale 9 di Arduino, mentre il suo pin **Echo** (secondo da destra) va collegato al pin digitale 10 di Arduino. Il codice del programma che fa ciò che vogliamo è il seguente:

Come si può ormai immaginare, il programma inizia con la solita dichiarazione delle variabili. La variabile **triggerPort** indica il pin di Arduino cui collegato il pin **Trig** del sensore, così come la **echoPort** indica il pin di Arduino cui è collegato il pin **Echo** del sensore. La variabile **led** indica il pin cui è collegato il led: ricordiamoci che deve essere uno di quelli contrassegnati dal simbolo ~ sulla scheda Arduino.



Il sensore di distanza HCSR04 ha 4 pin da collegare ad Arduino

La funzione **setup** stavolta si occupa di impostare la modalità dei tre pin digitali, chiamando per tre volte la funzione **pinMode** che abbiamo già visto. Stavolta, però, due pin (quello del trigger del sensore e quello del led) hanno la modalità **OUTPUT**, mentre il pin dedicato all'echo del sensore ha

modalità **INPUT**. Infatti, il sensore funziona emettendo degli ultrasuoni grazie al proprio pin trigger, e poi ascolta il loro eco di ritorno inviando il segnale ad Arduino tramite il pin echo. Il principio fisico alla base è che più distante è un oggetto, maggiore è il tempo necessario affinché l'eco ritorni indietro e venga rilevato dal sensore: è un sonar, come quello delle navi o dei delfini.

Prima di concludere la funzione setup, provvediamo ad aprire una comunicazione sulla **porta seriale**, così potremo leggere dal computer il dato esatto di distanza dell'oggetto.

Inizia ora la funzione loop, qui scriveremo il codice che utilizza il sensore a ultrasuoni.

Il sensore a ultrasuoni
Quando si vuole realizzare qualcosa di interattivo ma non troppo invasivo, il sensore a ultrasuoni è una delle opzioni migliori. Noi ci siamo basati sull'HCSR04, uno dei più comuni ed economici, si può trovare su Ebay ed AliExpress per più o meno 2 euro. Misurando variazioni nella distanza il sensore può anche permetterci di capire se la persona posizionata davanti ad esso si stia muovendo, quindi possiamo sfruttarlo anche come sensore di movimento.

L'impulso ad ultrasuoni

Abbiamo detto che il sonar funziona inviando un impulso a ultrasuoni e ascoltandone l'eco. Dobbiamo quindi innanzitutto inviare un impulso: e lo faremo utilizzando la funzione **digitalWrite** per accendere brevemente il pin cui è collegato

il trigger del sensore.

Prima di tutto il sensore deve essere spento, quindi impostiamo il pin al valore **LOW**, ovvero 0 Volt. Poi accendiamo il sensore in modo che emetta un suono (nelle frequenze degli ultrasuoni) impostando il pin su **HIGH**, ovvero dandogli 5 Volt. Ci basta un impulso molto breve, quindi dopo avere atteso 10 microsecondi (cioè 0,01 millesimi di secondo) grazie alla funzione **delayMicroseconds**, possiamo spegnere di nuovo il pin che si occupa di far emettere il suono al sensore portandolo di nuovo agli 0 Volt del valore **LOW**. Insomma, si spegne l'emettitore, lo si accende per una frazione di secondo, e lo si spegne: questo è un impulso.

Ora dobbiamo misurare quanto tempo passa prima che arrivi l'eco dell'impulso. Possiamo farlo utilizzando la funzione **pulseIn**, specificando che deve rimanere in attesa sul pin echo del sensore finché non arriverà un impulso di tipo **HIGH**, ovvero un impulso da 5V (impulsi con voltaggio inferiore potrebbero essere dovuti a normali disturbi nel segnale). La funzione ci fornisce il tempo in millisecondi, quindi per esempio 3 secondi saranno rappresentati dal numero 3000. Possiamo registrare questo numero nella variabile **durata**, che dichiariamo in questa stessa riga di codice. La variabile è dichiarata come tipo **long**: si tratta di un numero intero molto lungo. Infatti, su un Arduino una variabile di tipo **int** arriva al massimo a contenere il numero **32768**, mentre un numero intero di tipo long può arrivare a **2147483647**. Visto che la durata dell'eco può essere un numero molto grande, se utilizzassimo una variabile di tipo int otterremmo un errore. Ci si potrebbe chiedere: perché allora non si utilizza direttamente il tipo long per tutte le variabili? Il fatto è che la memoria di Arduino è molto limitata, quindi è meglio non intasarla senza motivo, e usare **long** al posto di **int**

soltanto quando è davvero necessario.

Come abbiamo visto per il sensore di temperatura, anche in questo caso serve una semplice formula matematica per ottenere la distanza (che poi è il dato che ci interessa davvero) sulla base della durata dell'impulso. Si moltiplica per 0,034 e si divide per 2, come previsto dai produttori del sensore che stiamo utilizzando. E anche in questo caso la variabile **distanza** va dichiarata come tipo **long**, perché può essere un numero abbastanza grande.

Ora possiamo scrivere un messaggio sulla porta seriale, per comunicare al computer la distanza rilevata dal sensore.

Naturalmente dobbiamo tenere conto di un particolare: il sensore ha dei limiti, non può misurare la distanza di oggetti troppo lontani. Secondo i produttori, se il sensore impiega più di 38 secondi (ovvero **38000** millisecondi) per ottenere l'eco, significa che l'oggetto che si trova di fronte al sensore è troppo distante. Grazie ad un semplice **if** possiamo decidere di scrivere un messaggio che faccia sapere a chi sta controllando il monitor del computer che siamo fuori portata massima del sensore.

Continuando l'if con un **else**, possiamo dire ad Arduino che se, invece, il tempo trascorso per avere un impulso è inferiore ai 38000 millisecondi, la distanza rilevata è valida. Quindi possiamo scrivere la distanza, in centimetri, sulla porta seriale, affinché possa essere letta dal computer collegato ad Arduino tramite USB.

Mappare i numeri

È finalmente arrivato il momento di eseguire la dissolvenza sul led collegato ad Arduino. Noi vogliamo che la luminosità del led cambi in base alla durata dell'impulso (la quale è, come abbiamo detto, proporzionale alla distanza dell'oggetto più vicino). Però la variabile **durata** può avere un qualsiasi valore tra 0 e **38000**, mentre il led accetta soltanto valori di luminosità che variano tra 0 e **255**.

Poco male: abbiamo già visto la volta scorsa che per risolvere questo problema possiamo chiamare la funzione **map**, indicando la variabile che contiene il numero da tradurre e i due intervalli. Il risultato viene memorizzato in una nuova variabile di tipo **int** (del resto è comunque un numero piccolo) che chiamiamo **fadeValue**. Questa variabile conterrà, quindi, un numero da 0 a 255 a seconda della distanza del primo oggetto che si trova di fronte al sensore.

Possiamo concludere impostando il valore appena calcolato, memorizzato nella variabile **fadeValue**, come valore del pin cui è collegato il LED (identificato dalla variabile led che avevamo dichiarato all'inizio del programma). Per farlo utilizziamo la funzione **analogWrite**, che per l'appunto non si limita a scrivere "acceso" o "spento" come la **digitalWrite**, ma piuttosto permette di specificare un numero (da 0 a 255, proprio come quello appena calcolato) per indicare la luminosità desiderata del led. Ora il ciclo else è completato, lo chiudiamo con una parentesi graffa.

Prima di concludere definitivamente la funzione loop, con una ultima parentesi graffa chiusa, utilizziamo la funzione **delay**

per dire ad Arduino di aspettare **1000** millisecondi, ovvero 1 secondo, prima di ripetere la funzione (e dunque eseguire una nuova lettura della distanza con il sensore ad ultrasuoni).

Controllare anche i led da illuminazione
Arduino fornisce, tramite i suoi pin digitali, al massimo 5Volt e 0,2Watt, che sembrano pochi ma sono comunque sufficienti per accendere i classici led di segnalazione ed alcuni piccoli led da illuminazione che si trovano nei negozi di elettronica. Ma è comunque possibile utilizzare Arduino per accendere led molto più potenti, anche fino a 90Watt (ed una normale lampadina led domestica ha circa 10W), utilizzando un apposito Shield, ovvero un circuito stampato da montare sopra ad Arduino: https://www.sparkfun.com/products/10618.

4 Leggere l'ora da internet e segnalarla con una lancetta su un servomotore

In questo progetto affrontiamo due temi importanti: il movimento e internet. Arduino, infatti, può manipolare molto facilmente dei servomotori, e dunque si può utilizzare per applicazioni meccaniche di vario tipo, anche per costruire dei robot. Inoltre, Arduino può essere collegato a internet: esistono diversi metodi per farlo, a seconda della scheda che si sta utilizzando. Un Arduino Uno può essere collegato ad internet tramite lo shield Ethernet, acquistabile a parte sia in versione semplice che con PowerOverEthernet (Arduino verrebbe alimentato direttamente dal cavo ethernet). Questo è lo scenario più tipico, ed è quello su cui ci basiamo per il nostro esempio. In alternativa, si può ricorrere a un WemosD1, che è fondamentalmente un Arduino Uno con un chip wifi

integrato, abbastanza facile da programmare e economico (costa poco più di un Arduino Uno). Chiaramente, il WiFi va configurato usando le apposite funzioni indicando la password di accesso alla rete, cosa che non serve per le connessioni ethernet. Esiste anche l'ottimo Arduino Yun, che integra sia una porta ethernet che una antenna WiFi, ed ha una comoda interfaccia web per configurare la connessione, senza quindi la necessità di configurare il WiFi nel codice del proprio programma. Arduino Yun è il più semplice da utilizzare, ma è un po' costoso (circa 50 euro), mentre la coppia Arduino Uno + Ethernet shield è molto più economica (la versione ufficiale arriva al massimo a 40 euro, e si può comprare una riproduzione made in China per meno di 10 euro su AliExpress). Lo shield deve semplicemente essere montato sopra ad Arduino. Il servomotore, invece, va collegato ad Arduino in modo che il polo positivo (**rosso**) sia connesso al pin **5V**, mentre il negativo (**nero**) sia connesso al pin **GND** di Arduino. Infine, il pin del segnale del servomotore (tipicamente **bianco** o in un altro colore) va collegato ad uno dei pin digitali **PWM** di Arduino, quelli contrassegnati dal simbolo ~ che abbiamo già visto per la dissolvenza del led. Nel nostro progetto di prova, realizzeremo un orologio: per semplificare le cose, utilizzeremo il servomotore per muovere soltanto la lancetta delle ore, tralasciando quella dei minuti. Il codice è il seguente:

Sono necessarie tre librerie esterne: **SPI** ed **Ethernet** ci servono per utilizzare la connessione ethernet. Invece, **Servo** è utile per controllare il servomotore.

Proprio per utilizzare il servomotore si deve creare una variabile speciale, che chiamiamo **myservo**, di tipo **Servo**. Questa non è proprio una variabile, è un "oggetto". Gli **oggetti** sono degli elementi del programma che possono avere una serie di proprietà (variabili) e di funzioni tutte loro. Per esempio, l'oggetto **myservo** appena creato avrà tra le sue

proprietà la posizione del motore, mentre l'oggetto **client** avrà tra le sue funzioni "personali" una funzione che esegue la connessione a internet. Gli oggetti servono a risparmiare tempo e rendere la programmazione più intuitiva: lo vedremo tra poco.

Una scheda Arduino Uno con lo shield ethernet montato sopra di essa

La pagina web da leggere

Dichiariamo due variabili che ci permettono di impostare il pin cui è collegato il segnale del servomotore, ed il MAC dell'Ethernet shield. Ogni dispositivo ethernet ha infatti un codice MAC che lo identifica: possiamo scegliere qualsiasi cosa, anche se in teoria dovremmo indicare quello che è stampato sull'ethernet shield.

È importante non avere nella propria rete locale due dispositivi con lo stesso codice MAC, altrimenti il router non riesce a distinguerli. Quindi basta cambiare un numero (esadecimale) qualsiasi nel codice dei programmi che si scrivono.

Ora specifichiamo due informazioni: il **nome del server** che

vogliamo contattare, e la **pagina** che vogliamo leggere da esso. Per far leggere ad Arduino la pagina web http://codice-sorgente.it/UTCTime.php?zone=Europe/Rome, dobbiamo dividere il nome del server dal percorso della pagina, perché questo è richiesto dal protocollo HTTP. Questa pagina è un classico esempio di **API** web, ovvero una pagina web messa a disposizione di programmatori per ottenere informazioni varie. In particolare, questa pagina ci fornisce in tempo reale la data e l'ora per il fuso orario che abbiamo selezionato: nell'esempio scegliamo Roma, ma potremmo indicare Atene scrivendo Europe/Athens.

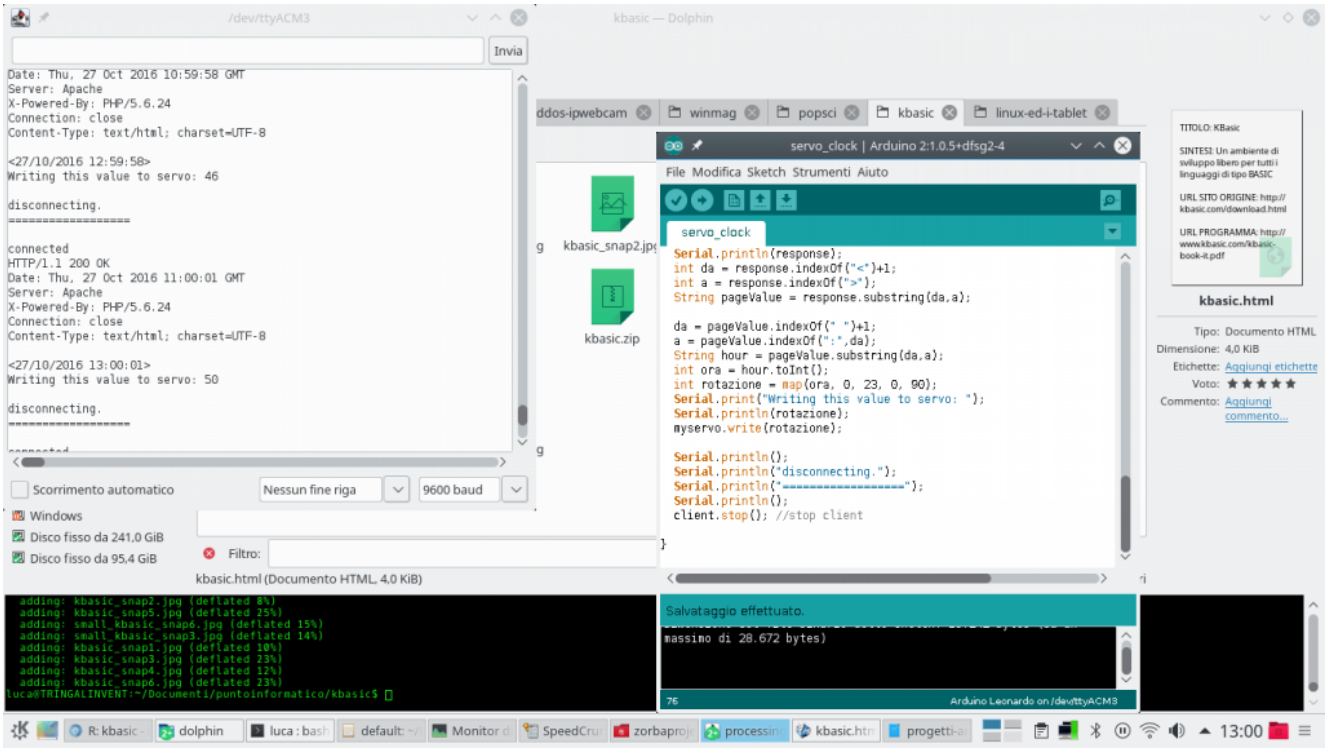
Altre API web

Abbiamo introdotto il concetto di API web, ovvero di una pagina web che contiene un semplice testo a disposizione dei programmatori. Nel nostro esempio ne utilizziamo una che fornisce l'ora attuale, ma esistono pagine web simili per qualsiasi cosa: uno tra i fornitori più importanti è Google, che offre la possibilità di eseguire ricerche con il suo motore, sfruttare i servizi di Maps, addirittura inviare email. Ma anche Twitter offre API con cui leggere od inviare tweet. Ne esistono migliaia, ed il sito migliore per trovarle è https://www.publicapis.com/. Molte di queste API richiedono un nome utente, che in genere è gratuito ma necessita di una iscrizione al sito web.

Se volete provare a realizzare una vostra pagina web che contenga l'ora attuale, come quella che sfruttiamo nel nostro esempio, dovete soltanto copiare questo file PHP:
https://pastebin.com/JeMf8p84
sul vostro server web (per esempio, Altervista ed Aruba supportano PHP).

La solita funzione **setup** comincia aprendo una comunicazione sulla porta seriale, così potremo leggere i messaggi di Arduino dal nostro computer, se vogliamo. Poi inizia con una

condizione **if**: questa ci permette di tentare la connessione al router, utilizzando la funzione **Ethernet.begin**, che richiede come argomento il codice **MAC** che avevamo scritto poco fa.



Con il monitor seriale di Arduino IDE possiamo leggere i messaggi che Arduino invia al nostro computer

La funzione **Ethernet.begin** fornisce una risposta numerica: il numero indica lo stato attuale della connessione, e se lo stato è **0**, significa che non c'è connessione. Quindi l'istruzione **if** può riconoscere questo numero e, nel caso sia **0**, prendere provvedimenti. In particolare, oltre a scrivere un messaggio di errore sulla **porta seriale**, così che si possa leggere dal computer collegato ad Arduino, se non c'è una connessione ethernet viene iniziato un **ciclo while infinito**: questo ciclo dura in eterno, impedendo qualsiasi altra operazione ad Arduino. Lo facciamo per evitare che il programma possa andare avanti se non c'è una connessione. Praticamente, Arduino rimane in attesa di essere spento. Un ciclo è una porzione di codice che viene eseguita più volte: nel caso di un ciclo while (che significa "mentre") il codice contenuto nel ciclo viene ripetuto finché la condizione posta

è vera (**true**). In questo caso il ciclo è sempre vero, perché come condizione abbiamo indicato proprio il valore **true**, e non contiene nessun codice, quindi il suo solo effetto è bloccare Arduino. Approfondiremo più avanti i cicli.

Prima di concludere la funzione **setup**, chiamiamo una funzione dell'oggetto **myservo**, che rappresenta il servomotore connesso ad Arduino. La funzione si chiama **attach** e serve a specificare che il nostro servomotore, d'ora in poi rappresentato in tutto e per tutto dal nome **myservo**, è attaccato al pin digitale di Arduino contraddistinto dal numero **3** (numero che avevamo indicato nella variabile **servopin**).

Connettersi al server web

La funzione **loop**, stavolta è molto semplice, perché deleghiamo buona parte del codice ad un'altra funzione: la funzione **sendGET**, che viene chiamata ogni 3 secondi.

In altre parole, la funzione **loop**, che viene eseguita continuamente, non fa altro che aspettare 3000 millisecondi, ovvero 3 secondi, e poi chiamare la funzione **sendGET**: è quest'ultima a fare tutto il lavoro, ed adesso dovremo scriverla.

Le parentesi graffe
Su sistemi Windows, con tastiera italiana, si può digitare una parentesi graffa premendo i tasti **AltGr + Shift + è** oppure **AltGr + Shift + +**. Infatti, i tasti **è** e **+** sono quelli che contengono anche le parentesi quadre. Quindi, premendo solo **AltGr + è** si ottiene la parentesi quadra, mentre premendo **AltGr + Shift + è** si ottiene la parentesi graffa. Su un sistema GNU/Linux, invece, si può scrivere la parentesi graffa

premendo i tasti **AltGr + 7** oppure **AltGr + 0**.

Questa è la nostra funzione **sendGET**, nella quale scriveremo il codice necessario per scaricare la pagina web con Arduino e analizzarla in modo da scoprire l'ora corrente.

Per cominciare si deve aprire una connessione HTTP con il server in cui si trova la pagina web che vogliamo scaricare. Le connessioni HTTP sono (in teoria) sempre eseguite sulla porta **80**, e il nome del server è memorizzato nella variabile **serverName** che avevamo scritto all'inizio del programma: queste due informazioni vengono consegnate alla funzione **connect** dell'oggetto **client**. Se la connessione avviene correttamente, la funzione **connect** fornisce il valore **true** (cioè vero), che viene riconosciuto da **if** e quindi si può procedere con il resto del programma. Come avevamo detto, l'oggetto **client** possiede alcune funzioni (scritte dagli autori di Arduino) che facilitano la connessione a dei server di vario tipo, e per utilizzare le varie funzioni basta utilizzare il punto (il simbolo **.**), posizionandolo dopo il nome dell'oggetto (ovvero la parola **client**).

Per esempio, un'altra funzione molto utile dell'oggetto **client** è **print** (e la sua simile **println**). Esattamente come le funzioni **print** e **println** dell'oggetto **Serial** ci permettono di inviare messaggi sul cavo USB, queste funzioni permettono l'invio di messaggi al server che abbiamo appena contattato. Per richiedere al server una pagina gli dobbiamo inviare un messaggio del tipo **GET pagina HTTP/1.0**. Visto che avevamo indicato la pagina all'inizio del programma, stiamo soltanto inserendo questa informazione nel messaggio con la funzione **print** prima di terminare il messaggio con **println**, esattamente come succede per i messaggi su cavo USB diretti al nostro computer (anch'essi vanno terminati con **println**, altrimenti

non vengono inviati).

È poi richiesto un messaggio ulteriore, che specifichi il nome del server da cui vogliamo prelevare la pagina, nella forma **Host: server**. Anche questo messaggio viene inviato come il precedente.

Ora la connessione è completata ed abbiamo richiesto la pagina al server, quindi il blocco **if** è terminato.
Naturalmente, possiamo aggiungere una condizione **else** per decidere cosa fare se la connessione al server non venisse portata a termine correttamente (ovvero se il risultato della funzione **client.connect** fosse **false**).

In questo caso basta scrivere sul monitor seriale un messaggio per l'eventuale computer collegato ad Arduino, specificando che la connessione è fallita. Il comando **return** termina immediatamente la funzione, impedendo che il programma possa proseguire se non c'è una connessione.

La risposta del server

Il blocco **if** e anche il suo **else** sono terminati. Quindi se il programma sta continuando significa che abbiamo eseguito una connessione al server e abbiamo richiesto una pagina web. Ora dobbiamo ottenere una risposta dal server, che memorizzeremo nella variabile **response**, di tipo **String**. Come abbiamo già detto, infatti, le **stringhe** sono un tipo di variabile che contiene un testo, e la risposta del server è proprio un testo.

La stringa **response** è inizialmente vuota, la riempiremo man mano.

Prima di cominciare a leggere la risposta del server dobbiamo assicurarci che ci stia dicendo qualcosa: potrebbe essere necessario qualche secondo prima che il server cominci a trasmettere. Un ciclo **while** risolve il problema: i cicli while vengono eseguiti continuamente finché una certa condizione è valida. Nel nostro caso, vogliamo due condizioni: una è che il client deve essere ancora connesso al server, cosa che possiamo verificare chiamando la funzione **client.connected**. L'altra condizione è che il client non sia disponibile, perché se non lo è significa che è occupato dalla risposta del server e quindi la potremo leggere. Possiamo sapere se il client è disponibile con la funzione **client.available**, la quale ci fornisce il valore **true** quando il client è disponibile. Però noi non vogliamo che questa condizione sia vera: la vogliamo falsa, così sapremo che il client non è disponibile. Possiamo negare la condizione utilizzando il **punto esclamativo**. In altre parole, la condizione **client.available** è vera quando il client è disponibile, mentre **!client.available** è vera quando il client non è disponibile. Abbiamo quindi le due condizioni, **client.connected** e **!client.available**. La domanda è: come possiamo unirle per ottenere una unica condizione? Semplice, basta utilizzare il simbolo **&&**, la doppia e commerciale, che rappresenta la congiunzione "e" ("and" in inglese). Quindi la condizione inserita tra le parentesi tonde del ciclo **while** è vera solo se il client è contemporaneamente connesso ma non disponibile. Significa che appena il client diventerà nuovamente disponibile oppure si disconnetterà, il ciclo while verrà fermato ed il programma potrà proseguire.

Ora, se ci troviamo in questo punto del programma, significa che il ciclo precedente si è interrotto, e il client ha quindi registrato tutta la risposta del server alla nostra richiesta di leggere la pagina web. Possiamo leggere la risposta una lettera alla volta utilizzando la funzione **client.read()**. La lettera viene memorizzata in una variabile di tipo **char**, ovvero un carattere, che viene poi aggiunta alla variabile

response. Il simbolo **+=**, infatti, dice ad Arduino che il carattere **c** va aggiunto alla fine dell'attuale stringa **response**. Se il carattere è "o" e la stringa è "cia", il risultato dell'operazione sarà la stringa "ciao".
Ovviamente, per leggere tutti i caratteri della risposta del server abbiamo bisogno di un ciclo che ripeta la lettura finché il client non ha più lettere da fornirci, un ciclo **while**. Per quante volte si deve ripetere il ciclo, cioè con quale condizione lo dobbiamo ripetere? Semplice: dobbiamo ripeterlo finché il client è connesso al server (perché è ovvio che se la connessione è caduta non ha più senso continuare a leggere), oppure finché il client è disponibile (perché se non è più disponibile alla lettura significa che la risposta del server è terminata e non c'è più niente da leggere). Stavolta abbiamo due condizioni, come nel ciclo precedente: una sarà **client.connected** e l'altra **client.available**, ma non le vogliamo necessariamente entrambe vere. Infatti, ci basta che una delle due sia vera per continuare a leggere. Insomma, l'una oppure l'altra. E il simbolo per esprimere la congiunzione "oppure" è **||**, cioè la doppia pipe (il simbolo che sulle tastiere italiane si trova sopra a \).

Se anche questo ciclo while è terminato, significa che tutta la risposta del server è ormai contenuta nella variabile **response**, quindi possiamo inviarla all'eventuale computer connesso ad Arduino tramite cavo USB.

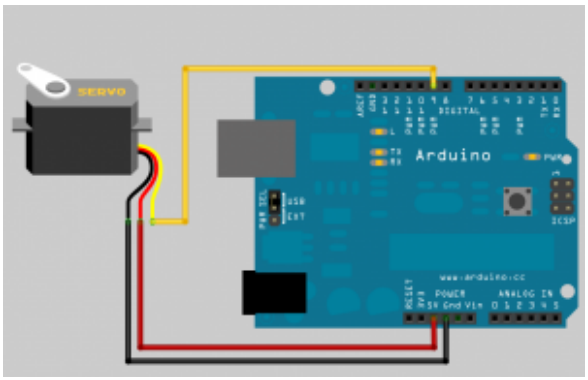
Estrarre l'informazione dalla pagina web

Ora possiamo cominciare a manipolare la risposta del server per estrarre le informazioni che ci interessano.

Innanzitutto, dobbiamo estrarre il contenuto della pagina web: se leggete la risposta completa del server, vi accorgerete che sono presenti anche molte informazioni accessorie che non ci interessano. Dobbiamo identificare il contenuto della pagina web: se provate a visitare la pagina http://codice-sorgente.it/UTCTime.php?zone=Europe/Rome noterete che il suo contenuto è qualcosa del tipo **<07/03/2019 21:33:56>**. Quindi, il suo contenuto è facilmente riconoscibile in quanto compreso tra il simbolo **<** e il simbolo **>**. Possiamo cercare il simbolo iniziale e quello finale all'interno della stringa **response** utilizzando la funzione **indexOf** tipica di ogni stringa, (funzione cui si accede con il solito simbolo **.**). Per esempio, scrivendo **response.indexOf("<")** ci viene fornita la posizione del simbolo **<** all'interno della stringa **response**. La posizione è un numero, per esempio potrebbe essere **42** se il simbolo **<** fosse il quarantaduesimo carattere dall'inizio del testo. Similmente, si può trovare la posizione del simbolo **>**, ed entrambe le posizioni si memorizzano in due variabili di tipo **int** (visto che sono numeri), che chiamiamo rispettivamente **da** e **a**. Da notare che alla prima posizione abbiamo sommato **1**, altrimenti sarebbe stato considerato anche il simbolo **<**, invece noi vogliamo tenere conto dei caratteri che si trovano dopo di esso. Utilizzando queste due posizioni, possiamo poi sfruttare la funzione **substring** della stessa **stringa** per ottenere tutto il testo compreso tra i due simboli. La funzione **substring** ci fornisce direttamente una stringa, che possiamo inserire in una nuova variabile chiamata **pageValue**.

L'attuale contenuto della stringa **pageValue** è qualcosa del tipo 07/03/2019 21:33:56. Noi siamo interessati soltanto al numero che rappresenta le ore, quindi possiamo ripetere la stessa procedura cambiando i valori delle variabili **da** ed **a**. In particolare, li dobbiamo cambiare affinché ci permettano di identificare il numero delle ore: questo è delimitato a sinistra da uno **spazio**, ed a destra dai **due punti**. Quindi,

utilizzando questi due simboli, la funzione **substring** riesce ad estrarre soltanto il numero delle ore, e inserirlo in una nuova variabile. Da notare che, anche se l'ora attuale è ai nostri occhi un numero, per il momento è ancora considerata un testo da parte di Arduino visto che finora lavoravamo con le stringhe. Possiamo trasformare questa informazione in un numero a tutti gli effetti grazie alla funzione **toInt** della stringa stessa, che traduce il testo "21" nel numero 21. Adesso, la variabile di tipo **int** (un numero intero) chiamata **ora** contiene l'orario attuale (solo le ore, non i minuti ed i secondi).



Collegare un servomotore ad Arduino è facile, basta ricordare che il pin digitale deve essere contrassegnato dal simbolo ~

Muovere il servomotore

Adesso abbiamo l'orario attuale, quindi possiamo spostare il servomotore in modo da direzionare la nostra lancetta. Per muovere un servomotore basta dargli una posizione utilizzando la sua funzione **write**. Quindi, nel nostro caso basta chiamare la funzione **myservo.write**.

Però c'è un particolare, le ore che ci fornisce il sito web

vanno da **0** a **23**, mentre le posizioni di un servomotore vanno da **0** a **180**. Infatti, un normale servomotore può ruotare di 180°: se chiamiamo la funzione **myservo.write(0)**, il motore rimarrà nella posizione iniziale, **se chiamiamo myservo.write(45)** verrà posizionato a 45° rispetto alla posizione iniziale. Il problema è: come traduciamo le ore del giorno in una serie di numeri compresa tra 0 e 180? Semplice, basta utilizzare la funzione **map**, che abbiamo già visto più volte. In questo modo le ore **23** diventerebbero **180**°, le ore **12** diventerebbero **45**°, e così via.

La funzione di mappatura

Abbiamo sempre usato la funzione map predefinita in Arduino. Ma, come esercizio di programmazione, possiamo anche pensare di scrivere una nostra versione della funzione. Che cos'è, quindi questa funzione? È una semplice proporzione, come quelle che si studiano a scuola:

Infatti, basta fare un rapporto tra i due range (quello del valore di partenza e quello del valore che si vuole ottenere). La funzione che abbiamo appena definito fornisce un numero con virgola (double), ma se preferiamo un numero intero ci basta modificare la definizione della funzione come tipo int.

Naturalmente, se vogliamo rappresentare le ore in un ciclo di 12 invece che di 24, basta dividere il numero per 12 tenendo non il quoziente ma il resto. Il resto della divisione si ottiene con l'operatore matematico %. Insomma, basta sostituire la variabile **ora** con la formula **(ora%12)**. In questo modo, le 7 rimangono le 7, mentre le 15 diventano le 3 (perché 15 diviso 12 ha il resto di 3).

Prima di completare la funzione, possiamo scrivere qualche

messaggio sul cavo USB per far sapere all'eventuale computer connesso ad Arduino che stiamo terminando la connessione al server che ha ospitato la pagina web. Poi la semplice funzione **stop** dell'oggetto **client** ci permette di concludere la connessione, e con la solita parentesi graffa chiudiamo anche la funzione, ed il programma è terminato.
Abbiamo detto che il nostro servomotore ruota di 180°: esistono anche altri servomotori modificati per poter eseguire una rotazione di 360°, chiamati **continuous rotation servo**, ma questi solitamente non si possono posizionare a un angolo preciso, piuttosto ruotano con velocità differenti. Ma sono casi particolari: un normale servomotore ruota di 180°, quindi la nostra lancetta dell'orologio si comporterebbe non tanto come un orologio normale, ma piuttosto come la lancetta di un contachilometri nelle automobili. È anche possibile cercare di modificare un servomotore da 180° per fare in modo che riesca a girare di 360°: http://www.instructables.com/id/How-to-modify-a-servo-motor-for-continuous-rotatio/?ALLSTEPS.
Ovviamente, per migliorare il nostro esempio, con un altro servomotore si può fare la stessa cosa per i minuti, ricordando però che sono 60 e non 24, quindi la funzione **map** va adattata di conseguenza.

Il codice sorgente

Potete trovare il codice sorgente dei vari programmi presentati in questo corso di introduzione alla programmazione con Arduino nel repository:
https://www.codice-sorgente.it/cgit/arduino-a-scuola.git/tree/
I file relativi a questa lezione sono, ovviamente, **dissolvenza-led-distanza-ultrasuoni.ino** e **ethernet-servomotore.ino**.

Corso di programmazione per le scuole con Arduino – PARTE 1

Quello che segue, e che proseguirà nelle prossime puntate, è un corso per assoluti principianti. È un corso per chi non ha mai scritto una linea di codice, ma vuole imparare a programmare. E come base per imparare i fondamenti della programmazione, abbiamo scelto C++ e Arduino. Perché è lo strumento più semplice da programmare e, soprattutto, concreto. Chi si avvicina alla programmazione ha bisogno di riscontri immediati, deve vedere immediatamente quale possa essere l'applicazione pratica di un concetto, altrimenti finirà per annoiarsi e rinunciare a studiare. Arduino è la soluzione perfetta per cominciare perché con un paio di righe di codice si possono ottenere risultati tangibili, e stimolare la curiosità per i capitoli successivi del corso di programmazione.

Per i docenti delle scuole (secondarie di primo e secondo grado): sentitevi pure liberi di utilizzare questa serie di articoli come "libro di testo".

Una introduzione per gli insegnanti

Nelle scuole italiane l'insegnamento dell'informatica è spesso trascurato, soprattutto per quanto riguarda la programmazione, che dovrebbe invece essere fondamentale per permettere agli

alunni lo sviluppo della logica e della capacità di risoluzione dei problemi.

Sicuramente, una parte di questo atteggiamento deriva dalla tendenza tutta italiana a considerare i computer soltanto come macchine da scrivere molto costose, invece che come strumenti davvero interattivi. è per questo motivo che dal ministero dell'istruzione, nonostante le riforme si rincorrano ogni 5 anni circa, non sono mai arrivate linee guida che suggeriscano come utilizzare i computer per l'insegnamento. E tutto viene lasciato nelle mani dei docenti, che per quanta buona volontà possano avere spesso non dispongono delle basi di informatica perché non hanno mai fatto parte della loro formazione.

Negli altri paesi non è così: la formazione digitale è considerata decisamente importante, e uno dei requisiti per diventare insegnante. Il rapporto del 2017 (http://ec.europa.eu/newsroom/document.cfm?doc_id=44390), a pagina 3, ci presenta tra le ultime posizioni sia in termini di competenze avanzate che come competenze basilari. A pagina 9 è persino possibile notare che, se consideriamo solo i lavoratori, le competenze digitali sono ancora minori, e ci fanno scendere ulteriormente in classifica, fino al terz'ultimo posto. E non è che nel nostro paese i computer siano davvero poco diffusi: tutte le scuole ne hanno a disposizione. Il problema è il modo in cui è pensata l'informatica a livello ministeriale, con i programmi che sono sempre troppo datati e troppo nozionistici, con poca attenzione all'aspetto pratico. Per cui è inevitabile che anche la formazione dei docenti stessi ne risenta, visto che la direzione verso cui si punta è sbagliata.

Nei rari istituti in cui si insegna la programmazione, solitamente scuole secondarie di secondo grado, lo si fa con strumenti e metodi obsoleti. Si utilizzano ancora Pascal e Fortran, linguaggi con cui uno studente oggi può fare ben poco di pratico e ai quali quindi non si affezionerà. Gli studenti vogliono qualcosa da poter esibire agli amici, e un programma

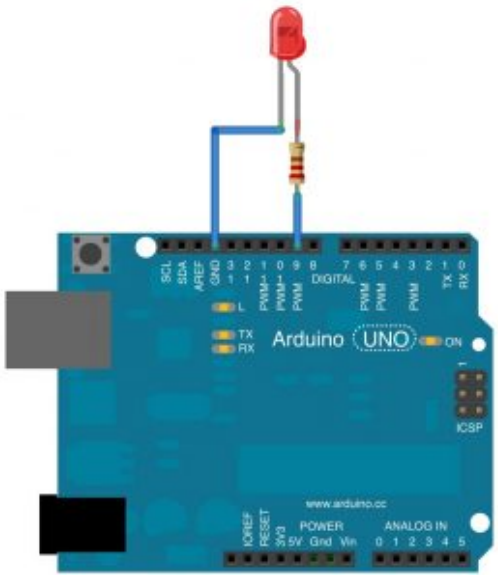
a riga di comando che somma un paio di numeri non è un gran trofeo. E non è nemmeno una cosa davvero utile, perché per la maggioranza delle operazioni che vengono descritte nei corsi di programmazione delle scuole esistono già altri programmi che risolvono il problema in modo molto più semplice e veloce. Siccome di solito le lezioni di programmazione sono tenute dai docenti di matematica, vengono descritte principalmente operazioni matematiche. Ma in realtà non ha senso: per imparare a fare la media di un elenco di numeri basta usare un foglio di calcolo, sarebbe una soluzione molto più adatta all'uso che nel mondo reale si fa dei computer. Se si vuole insegnare la programmazione bisogna prima di tutto spiegare il senso stesso della programmazione, cioè il motivo per cui valga la pena imparare come scrivere programmi. È un problema comune a molte discipline: gli studenti si annoiano sempre a realizzare riassunti nei compiti di italiano, e questo perché nessuno spiega loro qual è il senso stesso del riassunto (cioè imparare a estrarre informazioni da un testo e rielaborarle in modo proprio). Nessuno, studenti adolescenti in particolare, sarà mai ben disposto a fare qualcosa di cui non capisce l'utilità.

Procurarsi un arduino

Arduino è il minicomputer più diffuso tra gli artisti che vogliono rendere interattive le loro creazioni (è il motivo per cui venne creato in primo luogo), tra gli appassionati di modellismo che realizzano droni, ed anche tra i progettisti di dispositivi intelligenti (in particolare per i dispositivi Internet of Things). Ma è anche molto utilizzato nella scuole, soprattutto nei paesi del Nord Europa, per avvicinare i bambini alla programmazione. Arduino ha infatti il pregio di essere estremamente semplice da programmare, e avere tante applicazioni pratiche capaci di stimolare l'interesse dei neofiti. In Italia non è ancora molto diffuso a questo scopo, ma abbiamo pensato di proporvi alcuni progetti interessanti

con cui avvicinare alla programmazione i vostri figli e tutti gli amici che vorrebbero cominciare a scrivere codice ma si annoiano con i corsi teorici tradizionali. Del resto, una volta molti ragazzi diventavano programmatori perché attratti dalla possibilità di inventare un videogioco: oggi l'internet of things e la possibilità di costruire oggetti intelligenti, unendo la programmazione al bricolage, possono fungere da motivazione per avvicinarsi alla programmazione. Chi vuole procurarsi un Arduino originale può trovarlo su Amazon: per le scuole, la soluzione migliore è rappresentata da Farnell (https://it.farnell.com/arduino/a000066/arduino-uno-evaluation-board/dp/2075382), che permette pagamenti tramite il sistema MEPA contattando il servizio clienti per una quotazione personalizzata. In alternativa, un docente può semplicemente mettere dei fondi di tasca propria (o farli raccogliere dal rappresentante dei genitori) e comprare dei cloni di Arduino Uno su AliExpress (https://it.aliexpress.com/w/wholesale-arduino-uno-r3.html): costano mediamente 2,80 euro l'uno, quindi se si hanno 20 studenti bastano 56 euro per garantire a tutti un Arduino Uno. Tra l'altro, gli esempi che proponiamo possono essere realizzati anche usando un clone di Arduino Nano, che costa appena 1,70 euro. Così se uno studente brucia un paio di schede mandandole in corto circuito non è un grave danno economico. Se poi non si vuole dare una scheda a ciascuno studente, ma dividere la classe in gruppi di lavoro, è possibile permettere la sperimentazione sul sito TinkerCAD: https://www.tinkercad.com/circuits. Si tratta di un simulatore completamente gratuito, che permette di disegnare un circuito con la classica breadboard virtuale e testare il codice. È quindi perfetto per permettere agli studenti di provare sia il circuito che il codice del programma prima di caricarlo sul un vero Arduino e vedere come funziona. Chiaramente, oltre ad Arduino sono necessari anche alcuni componenti, come LED, sensori, buzzer, eccetera: i vari siti che abbiamo citato (Amazon, Farnell, AliExpress) offrono tutto il materiale di cui si può avere bisogno. Ci si può chiedere perché puntare su

Arduino, piuttosto che su PLC Siemens che costano decine di euro ciascuno: il fatto è che Arduino costa molto meno, ha un'ottima documentazione, e se si compra la scheda originale invece dei cloni si ha già una certificazione CE (http://web.archive.org/web/20170320180212/https://www.arduino.cc/en/Main/warranty). Questo significa che gli studenti che imparano a usare Arduino a scuola potranno anche utilizzarlo in seguito nella loro carriera lavorativa. Per quanto i PLC siano al momento più diffusi nei macchinari industriali, infatti, Arduino è certamente il modo migliore per iniziare, e molte aziende stanno cominciando a puntare soprattutto sui modelli più piccoli (Arduino Nano e Micro) per le applicazioni IoT. E se si deve cominciare a imparare la programmazione, è sempre meglio bruciare un piccolo Arduino da 2 euro, piuttosto che un PLC da 200 euro.



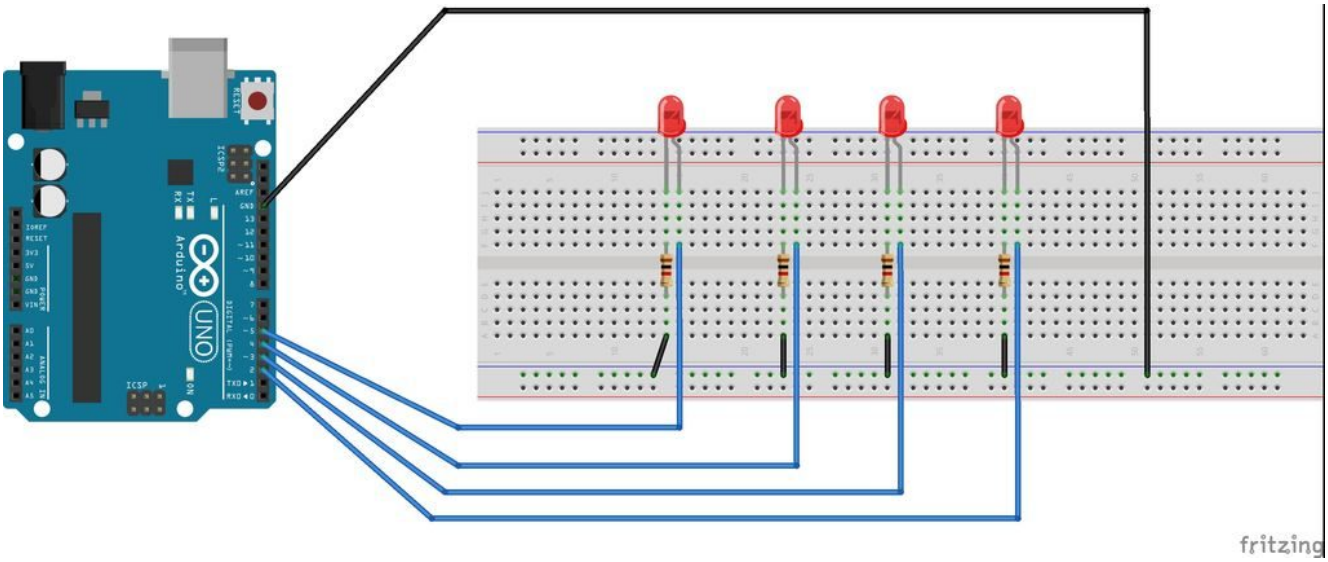
Un semplice Arduino UNO con un LED che può essere acceso o spento con un programma

Ottenuto un Arduino, lo si può programmare con l'Arduino IDE, che si può recuperare dalla pagina https://www.arduino.cc/en/Main/Software. È importante non confondersi con l'Arduino Web Editor, che è un'altra cosa.

L’Arduino IDE è un semplice programma gratuito e open source installabile su tutti i sistemi operativi, che permette di scrivere il codice dei propri programmi e caricarlo su una qualsiasi scheda Arduino (o derivate).

1 Una scala di led che si accendono in sequenza

Per cominciare, realizziamo un progetto semplice: una serie di led (Light Emissing Diode, piccole lampadine per circuiti elettrici a basso consumo) che si accenda a seconda della posizione di un potenziometro. In poche parole, avremo una rotella che si potrà girare per accendere da 1 a 5 lampadine. Un potenziometro è infatti una resistenza variabile, tipicamente una rotella od una leva che possiamo spostare per modificare la resistenza (come quelle con cui si cambia il volume sugli impianti stereo). Arduino è in grado di leggere un valore numerico in funzione della resistenza: il valore è compreso tra 0 e 1023, e noi possiamo scrivere un programma che riconosce questo numero ed in base alla sua posizione nell’intervallo 0-1023 accende 5 led.



Con una breadboard è facile collegare tanti led senza bisogno di saldare nulla

Per esempio, significa che se il potenziometro è a 0 tutti i led saranno spenti, se è a 1023 saranno tutti accesi, e se è a 200 soltanto due led saranno accesi. Per poter leggere il numero, il potenziometro va collegato ad uno dei pin analogici, mentre i led devono essere collegati a dei pin digitali, così potremo tenerli accesi o spenti. Il pin di sinistra del potenziometro va collegato al GND di Arduino, il pin centrale va connesso ad uno dei pin analogici, ed il pin di destra va connesso ai 5V di Arduino. Il pin lungo dei led va collegato ad uno dei pin digitali di Arduino, mentre il pin corto di ogni led va connesso al GND di Arduino. Il codice, che possiamo scrivere direttamente nell'Arduino IDE, è il seguente:

Prima di tutto definiamo alcune variabili: sono delle semplici parole a cui viene associato un valore di qualche tipo. Per esempio, una variabile di tipo **int** contiene un numero intero (mentre i numeri con la virgola sono di tipo **double**). La variabile chiamata **potPin** è quella che indica il pin (analogico) di Arduino cui abbiamo collegato il potenziometro: nell'esempio, si tratta del pin numero 2. Similmente, le varie variabili **ledPin** indicano i pin (digitali) di Arduino cui abbiamo collegato i led: ce ne sono 5 perché ciascuna di esse indica il pin di uno dei 5 led del nostro progetto. In pratica, abbiamo collegato ad Arduino 5 diversi led, dal pin 3 al 7. Il nome delle variabili è chiaramente a nostra discrezione, avremmo potuto chiamarle "arancia", "mela", e "banana", e si dichiarano sempre nella forma TIPO NOME = VALORE. L'assegnazione del primo valore non è fondamentale, ma è buona norma, e in questo caso si parla di "inizializzazione". Ora, definiamo la nostra prima funzione:

La funzione setup è una delle due funzioni standard di arduino (l'altra è loop). Tutto il codice contenuto dentro questa funzione verrà eseguito una sola volta, ovvero all'avvio di arduino (appena viene acceso). Una funzione, di fatto, non è

altro che un blocco di codice, un po’ come un capitolo all’interno di un libro. Ce ne sono molte già pronte all’uso, il cui codice è già stato scritto da qualcun altro, ma se vogliamo possiamo anche scrivere noi delle funzioni. Quando scriviamo una funzione dobbiamo indicare il suo tipo (void è un tipo molto comune, perché significa che la funzione fa qualcosa ma poi non fornisce un risultato che possa essere registrato dentro una variabile) ed il suo nome, cominciando poi a scrivere il suo codice con la parentesi graffa aperta. I tipi delle funzioni sono gli stessi delle variabili, perché sono tipi di dato generici. Per esempio, una funzione dichiarata come **int** potrà fornire un risultato che è un numero intero. Ma per ora ci bastano le funzioni che svolgono delle operazioni senza però fornire un risultato in termini numerici. Per esempio, nella funzione **setup** del nostro programma cominciamo a scrivere questo codice:

Qui stiamo usando la funzione (che non dobbiamo scrivere noi, è integrata in Arduino) chiamata **pinMode**, ed è anch’essa una funzione **void**, cioè una che non fornisce un risultato numerico ma si limita a fare cose. La funzione ci permette di abilitare dei ping per un segnale elettrico di output. I pin digitali di Arduino possono infatti avere due diverse modalità di funzionamento: INPUT ed OUTPUT. Siccome i nostri pin sono collegati a dei led, saranno ovviamente pin di output, perché non ci servono per fornire ad Arduino dei dati, ma piuttosto per fornire una informazione da Arduino ai led stessi (cioè, dobbiamo dire ai LED se debbano rimanere accesi o spenti). Quindi utilizzando la funzione **pinMode** possiamo impostare la modalità OUTPUT a tutti i vari pin dei led connessi ad Arduino. Il bello delle funzioni è proprio questo: quando una funzione è stata scritta, poi può essere chiamata ogni volta che vogliamo, eventualmente anche con degli argomenti (cioè informazioni utili alla procedura desiderata, come in questo caso il numero del pin e la modalità). Questo ci risparmia di dover scrivere ogni volta tutto il codice, scrivendo quindi

solo una riga. Esistono molte funzioni già presenti in Arduino, quindi non abbiamo bisogno di scriverle di nostro pugno, possiamo direttamente chiamarle.

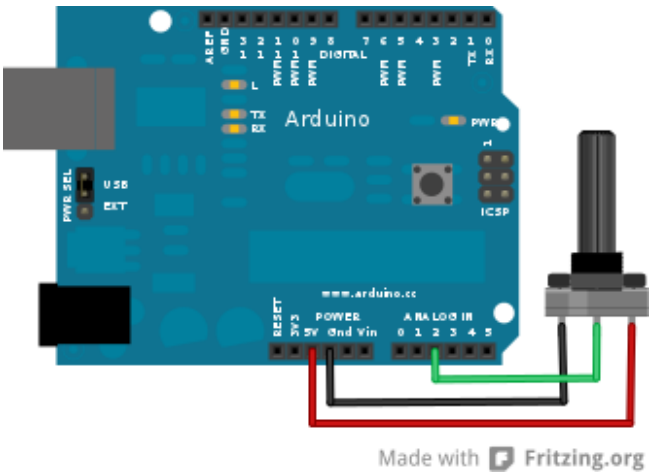
Il nostro programma è molto semplice, quindi possiamo chiudere qui la funzione setup, utilizzando una parentesi graffa chiusa. Riassumendo, la nostra funzione **setup**, eseguita automaticamente una sola volta all'accensione di Arduino, si limita a chiamare a sua volta la funzione **pinMode** per impostare i pin dei LED come output invece che input.

La funzione loop

Ora cominciamo a scrivere l'altra funzione standard di Arduino, chiamata loop.

Il codice contenuto dentro questa funzione verrà ripetuto all'infinito, finché Arduino non verrà spento. Quindi, è il codice che fa effettivamente qualcosa.

Definiamo una nuova variabile chiamata **val**, il cui valore iniziale sia 0 (per convenzione si indica sempre un valore iniziale pari a 0 per tutte le variabili, ma non è obbligatorio). Poi assegniamole il valore fornito dalla funzione **analogRead**. Questa è la funzione che legge il segnale del pin analogico che indichiamo: nel nostro caso abbiamo indicato il pin analogico cui è connesso il potenziometro, quindi la variabile val conterrà il numero, compreso tra 0 e 1023, relativo alla posizione del potenziometro.



Collegare un potenziometro a Arduino è facile

Per il momento, abbiamo solo letto la posizione del potenziometro, quindi sappiamo come è posizionata la rotella (per esempio, 0 sarà tutto a sinistra, 1023 tutto a destra, 500 circa a metà).

Ora la variabile **val** contiene un numero compreso tra 0 e 1023, ma per noi questo è scomodo: abbiamo 5 led, quindi ci farebbe comodo ricondurre il numero registrato dal potenziometro ad un intervallo compreso tra 0 e 5. Possiamo farlo chiamando la funzione **map**, che si occupa proprio di mappare (cioè tradurre) una variabile da un intervallo ad un altro. Nel caso specifico, chiediamo alla funzione map di mappare la variabile val dall'intervallo che da va 0 a 1023 sull'intervallo che va da 0 a 5. Il risultato di questa traduzione verrà memorizzato nella variabile **mappedval**, che abbiamo appena creato (con la riga int mappedval). Vale a dire che invece di avere un numero compreso tra 0 e 1023, per indicare la posizione della rotella, ne avremo uno compreso tra 0 e 5. Questa cosa si può fare perché la funzione map (che è sempre integrata in Arduino) non è una **void**, ma una **int**, e quindi ci fornisce un risultato numerico sotto forma di numero intero. Questo risultato può essere assegnato a una variabile semplicemente con il simbolo di uguaglianza.

Ora dobbiamo cominciare ad accendere i vari led: ovviamente, la loro accensione dipenderà dal valore appena ottenuto, che è memorizzato nella variabile **mappedval**. Secondo la logica del nostro progetto, il primo led si accenderà se il valore è superiore a 0, il secondo si accenderà se il valore è superiore ad 1, e così via (ricordiamo che lavoriamo con numeri interi, quindi non esistono valori come 0,3 o 1,5, vengono automaticamente arrotondati al numero intero più vicino). Possiamo ottenere questo risultato con un blocco if: i blocchi sono piccole porzioni di codice che hanno una qualche forma di controllo per la loro esecuzione. Per esempio i blocchi if (che in inglese significa il condizionale "se") sono porzioni di codice che vengono eseguite solo se una certa condizione è valida, mentre in caso contrario vengono completamente ignorate. Scrivendo **if (mappedval>0)**, tutto il codice che indichiamo dopo la parentesi graffa viene eseguito soltanto se la variabile **mappedval** ha un valore maggiore di 0.

Il codice che vogliamo eseguire in tale situazione è ovviamente molto semplice: vogliamo solo accendere il primo led. Per accendere un led con Arduino basta dare un valore alto (HIGH, ovvero 5 Volt) al pin cui il led è collegato, in questo caso ledPin1. Il valore può essere assegnato al pin chiamando la funzione **digitalWrite**.

Analogico o digitale?
La differenza fondamentale tra i due tipi di pin presenti su Arduino è che il primo può gestire segnali analogici, ed il secondo segnali digitali. Ma cosa significa questo, in breve? In pratica, un segnale analogico può avere tutta una serie di valori che, solitamente, spaziano dallo zero all'uno (ad esempio 0,4). Un segnale digitale, invece, può essere solamente 0 oppure 1. Questo non significa che uno sia meglio dell'altro: dipende tutto da ciò che dobbiamo fare. Se

vogliamo semplicemente avere (o dare) una informazione del tipo si/no un pin digitale è perfetto. Se, invece, vogliamo utilizzare una diversa scala di valori, ci conviene utilizzare un segnale analogico. Quest'ultimo è, dunque, il migliore per l'uso di sensori ambientali: esistono, comunque, dei sensori digitali, anche se sono più rari. Per quanto riguarda Arduino, i pin digitali sono 12, mentre quelli analogici sono 6. Inoltre è necessario inizializzare solo i pin digitali (scrivendo nel programma la riga pinMode(10, OUTPUT); per specificare che il pin 10 deve essere usato in output).
Chiudiamo il blocco if e continuiamo con il codice:

Uno dei lati interessanti di un blocco if è che quando lo terminiamo, con la parentesi graffa, possiamo indicare un codice alternativo con la parola **else** (che significa "altrimenti"). Quindi, se la la condizione specificata tra parentesi tonde (cioè mappedval>0) è vera, viene eseguito il codice compreso tra le prime parentesi graffe. Altrimenti, se tale condizione non è vera, viene eseguito il codice presente tra le parentesi graffe immediatamente dopo la parola **else**.

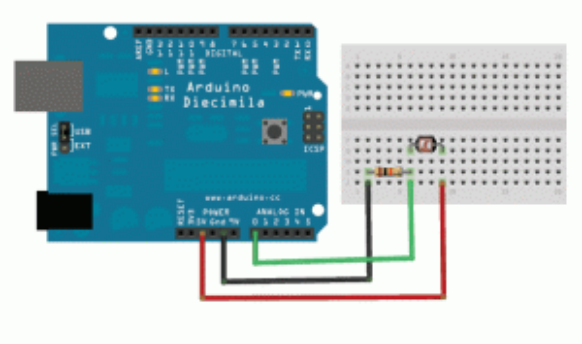
Naturalmente, noi vogliamo che se la condizione del ciclo if non è vera il led sia spento. Per farlo basta scrivere il valore basso (LOW, pari a 0 Volt) sul pin del led. Poi possiamo chiudere anche la parentesi graffa di else.

Si ripete per gli altri led

Il codice per il primo led è scritto, occupiamoci ora del secondo:

Similmente, costruiamo un ciclo if-else per il secondo led, identificato dal pin digitale ledPin2, che deve essere acceso (HIGH) solo se la variabile **mappedval** è maggiore di 1, e spento (LOW) in tutte le altre occasioni.

Si ripete lo stesso tipo di ciclo adattandolo agli altri tre led, così anch'essi possono essere accesi e spenti a seconda della posizione della rotella del potenziometro. Intuitivo, vero? Poi si può chiudere la funzione loop, che è quella di cui stavamo scrivendo il codice finora, con una parentesi graffa. Il programma è completo.



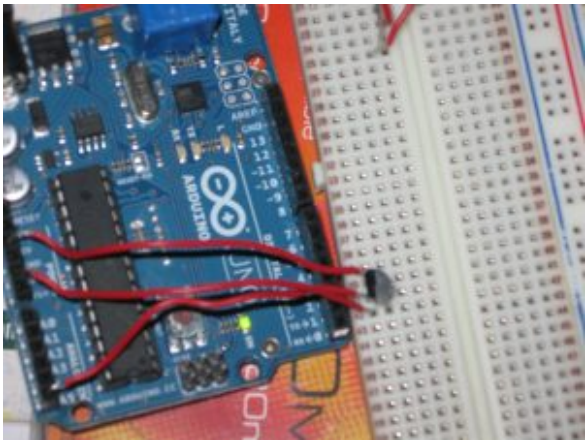
Il potenziometro può essere facilmente sostituito con una fotoresistenza

Volendo, si può anche sostituire il potenziometro con una fotoresistenza: sono dei semplici sensori di luminosità. Una fotoresistenza ha due pin perfettamente identici: uno va collegato al pin 5V di Arduino, l'altro al pin analogico 2. Quello collegato al pin analogico 2 va anche collegato, con una semplice resistenza da 10kOhm, al pin GND di Arduino. Similmente, si può sostituire il potenziometro con qualsiasi altro sensore capace di dare un segnale analogico, per esempio anche un microfono, con il quale si otterrebbe un visualizzatore di volume audio (i led si accenderebbero sempre di più man mano che si parla più forte al microfono). Serve solo un po' di fantasia.

2 Utilizzare un sensore LM35

per misurare la temperatura

Il potenziometro dell’esempio precedente è un sensore, perché permette di fornire dati ad Arduino. Esistono diversi altri sensori, che misurano informazioni relative all’ambiente: temperatura, pressione ed umidità, per esempio. Uno dei sensori più semplici da utilizzare è chiamato LM35, e misura la temperatura di una stanza. E possiamo utilizzare Arduino per leggerla, inviando il valore esatto al nostro computer attraverso il cavo USB. Cerchiamo innanzitutto di capire come funziona il sensore di temperatura LM35: contrariamente a quanto si potrebbe pensare, il sensore non ci fornisce direttamente la temperatura. Ci fornisce il solito valore compreso tra 0 e 1023, e questo valore è proporzionale alla temperatura. Con una semplice formula matematica siamo in grado di risalire alla temperatura corretta con una precisione di mezzo grado: è sufficiente moltiplicare il numero fornito dall’LM35 per circa 0,5 (esattamente per 500/1023=**0,488**, come indicato dai produttori del sensore) per ottenere la temperatura in gradi Celsius. È poi ovviamente possibile applicarle una serie di trasformazioni per cambiare scala: se per qualche motivo la volessimo in scala assoluta (i cosiddetti gradi Kelvin del sistema internazionale di misure) basterà sommare al valore ottenuto il numero 273,15. Il sensore andrà collegato ad Arduino nel seguente modo: guardando la scritta sul lato piatto, il pin a sinistra va collegato alla alimentazione a 5V, quello centrale va collegato al pin analogico 1 di Arduino, ed infine quello più a destra va collegato con il pin GND di Arduino. Se preferiamo rendere impermeabile il sensore, in modo da poterlo anche mettere a contatto con liquidi, basta ricoprirlo con della pellicola trasparente per alimenti.



Per collegare un sensore di temperatura a Arduino bastano tre cavi e una breadboard

Tenete comunque presente che, anche se il sensore può sopportare temperature che variano dai -55 °C ai 140 °C, potrebbe non essere una buona idea sottoporlo a situazioni critiche. Ad esempio, non immergetelo nella pentola dell'acqua per vedere quando bolle. Potete però appoggiarlo ad un cubetto di ghiaccio per vedere quando fonde. Il codice è relativamente semplice:

Cominciamo anche questo programma dichiarando due variabili: una è un **numero intero**, si chiama **pin**, e rappresenta il pin analogico di Arduino cui abbiamo collegato il sensore LM35, nel nostro caso il pin numero 1. L'altra è di tipo **double** perché si tratta di un **numero con virgola**, si chiama temp e rappresenterà la temperatura rilevata dal sensore (per ora la impostiamo uguale a 0.0, non dimentichiamo che con Arduino si utilizza la notazione internazionale che prevede il punto al posto della virgola per le cifre decimali). Cominciamo subito la funzione setup:

La funzione setup (eseguita una volta sola all'avvio di Arduino) non fa altro che avviare la comunicazione tramite porta seriale, con una velocità (bitrate) di 9600 baud, che è lo standard più comune. La porta seriale ovviamente è la porta

USB (Universal Serial Bus), tramite la quale potremo poi leggere i messaggi di Arduino con un apposito terminale dal nostro computer (il Monitor seriale dell'IDE di Arduino), al quale abbiamo collegato Arduino tramite il cavo USB.

La funzione loop, invece, (che è quella eseguita ripetutamente) legge il valore fornito dal sensore chiamando la funzione **analogRead**, che abbiamo già visto, moltiplicando immediatamente questo numero per il valore correttivo, ovvero 500/1023, assegnando il risultato dell'operazione alla variabile temp. In poche parole, abbiamo chiesto ad Arduino di calcolare il risultato di una semplice equazione: il numero fornito dal sensore viene moltiplicato per 500 e diviso per 1023, ed il risultato diventa il valore della variabile temp. Se al posto delle funzione **analogRead** avessimo scritto X ed al posto di temp avessimo scritto Y, l'avreste riconosciuta subito come una banale equazione di primo grado.

Ora possiamo scrivere dei messaggi al computer, il quale li riceverà sulla porta USB. I messaggi sono ovviamente dei testi (che in programmazione sono chiamati **stringhe**, e tipicamente delimitati dalle virgolette "), e possono essere inviati con la funzione **Serial.print**. Ogni messaggio è costituito da una riga, che termina quando chiamiamo la funzione **Serial.println()**. Possiamo quindi scrivere un messaggio composto dalle parole **Stanza1:** seguite dal valore della temperatura memorizzato nella variabile **temp** (che è un numero, ma viene automaticamente tradotto da Arduino in una stringa di testo) ed infine dal simbolo dei gradi Celsius. Il messaggio viene terminato con l'indicazione del termine della riga (**ln** in **println** significa **line new**, ovvero nuova riga).

L'ultima riga di codice della funzione loop dice ad Arduino di attendere 1 secondo (1000 millesimi di secondo, si ragiona sempre in millesimi di secondo) prima di eseguire nuovamente la funzione: questo significa che la temperatura verrà letta

una volta al secondo. Possiamo cambiare questo valore se vogliamo che la temperatura sia letta più spesso o più raramente.

I pin del sensore LM35 sono facili da riconoscere: guardando il lato piatto, da sinistra abbiamo il pin 5V (positivo), quello analogico, e il pin GND (negativo)

Dopo avere caricato lo sketch su Arduino, noterete che il piccolo led TX della scheda lampeggia. A questo punto si può cliccare sul menù **Strumenti/Monitor seriale** dell'Arduino IDE: apparirà una nuova finestra in stile prompt dei comandi con una serie di scritte del tipo "Stanza 1: 20.00°C", una riga per ogni secondo.