Corso di programmazione per le scuole con Arduino – PARTE 3

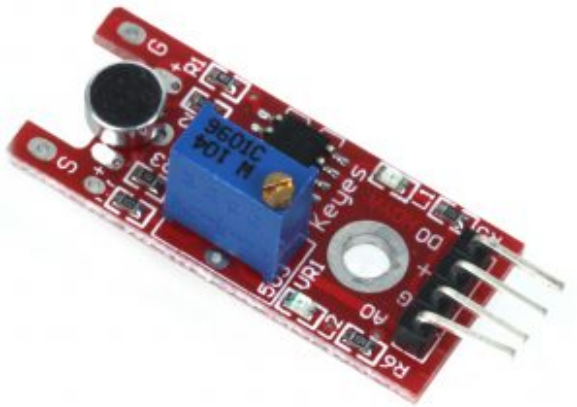
Nella scorsa puntata di questo corso abbiamo presentato una serie di esempi per imparare a programmare con Arduino partendo da zero, rivolti soprattutto a studenti, insegnanti, designer, e altre persone che non sono già abituate a programmare. Vogliamo ora proseguire con degli altri esempi, un po' più 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.

Table of Contents

- 5 Accendere un relay con un microfono
- Distinguere un suono dal rumore di fondo
- 6 Un allarme sonoro collegato ad una porta
- La melodia da suonare
- Solo se il pulsante non è premuto

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?

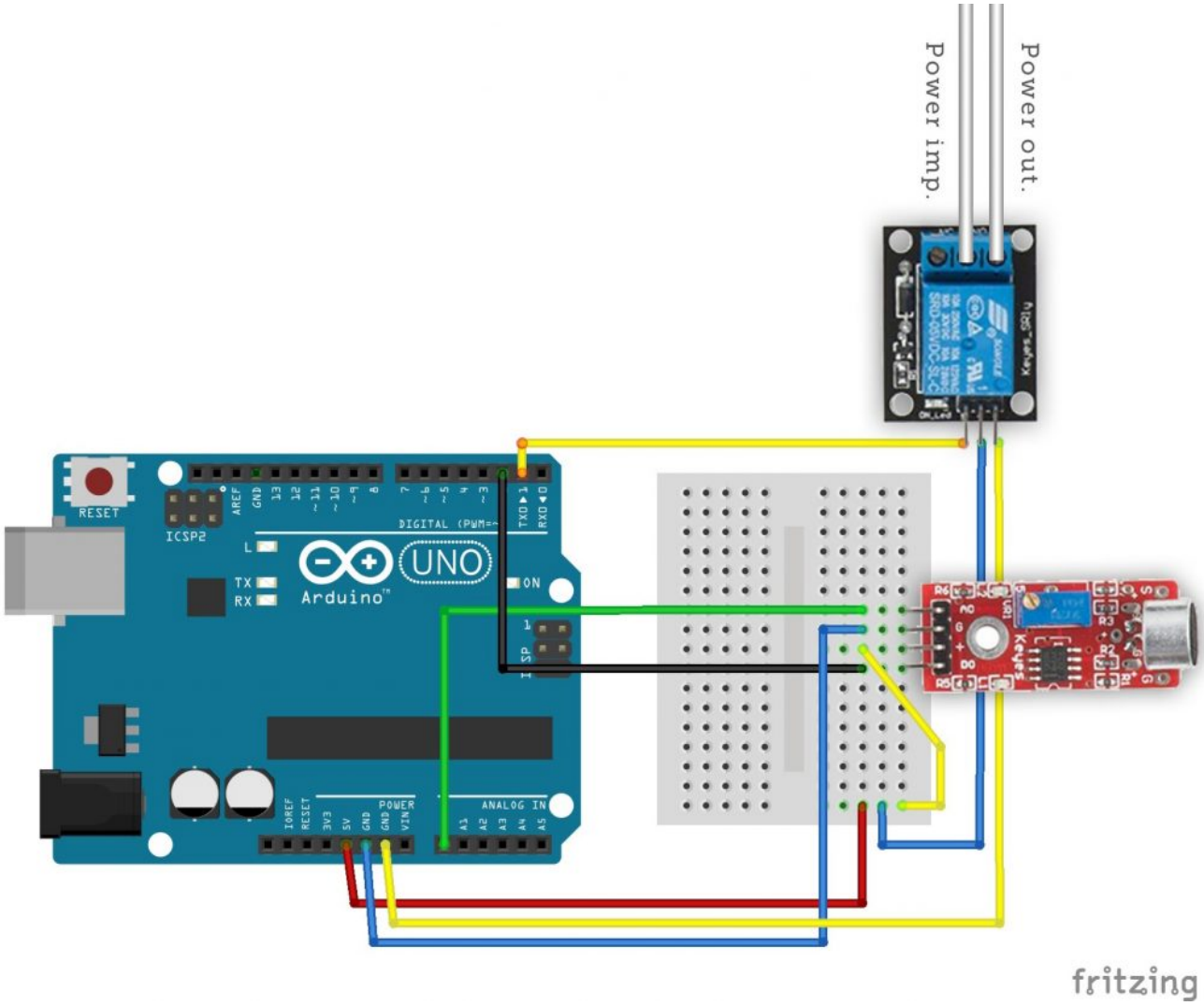
Negli esempi proponiamo dei sensori e degli altri componenti specifici. Mentre un relay si può acquistare in qualsiasi negozio di elettronica, ed anche un pulsante od un buzzer, il microfono e la scheda con i vari sensori infrarossi sono più

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

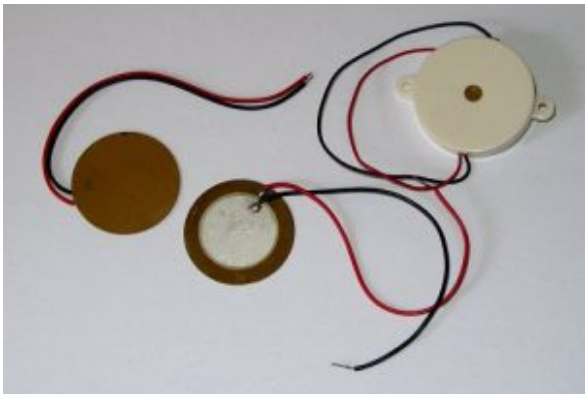
Per questo scopo, in realtà andrebbe bene anche un pulsante: lo si posiziona tra la porta e il muro, così finché la porta è chiusa il pulsante è premuto, mentre appena si apre il pulsante non sarà più premuto. Naturalmente, un pulsante può essere costituito con due pezzi di alluminio (anche quello da 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

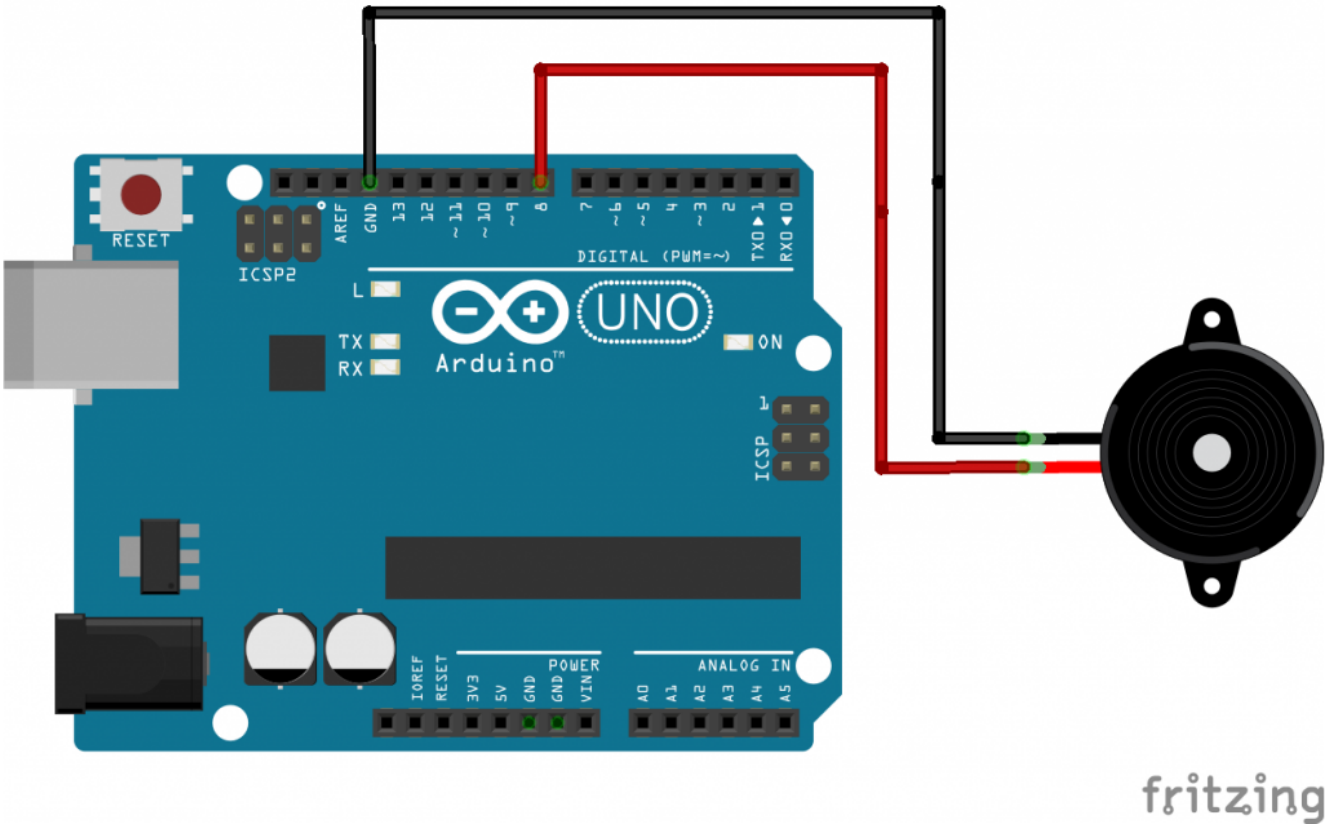
frequenze audio udibili da un essere umano, ed anche alcuni ultrasuoni udibili soltanto da altri animali (come i cani). Le note musicali non sono altro che specifiche frequenze. Le note sono in totale 12, considerando anche i bemolle ed i diesis, e vengono divise in 8 o 9 ottave. Un pianoforte, comunque, 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

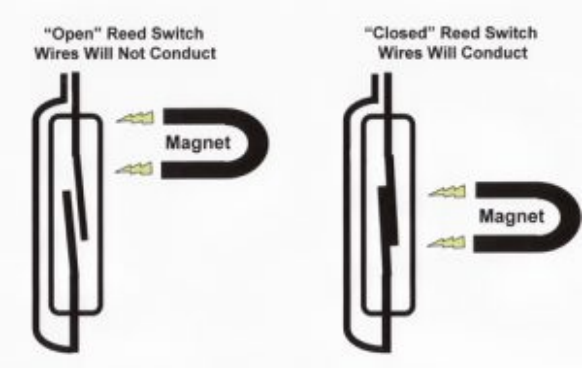
Un pulsante non è altro che un contatto di qualche tipo che ha due posizioni: aperto o chiuso. Quando il contatto è aperto non c'è passaggio di corrente, quando il contatto è chiuso la corrente passa. Un pulsante si costruisce facilmente con Arduino: basta inserire una resistenza da 10KOhm nel pin 5V, 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**.

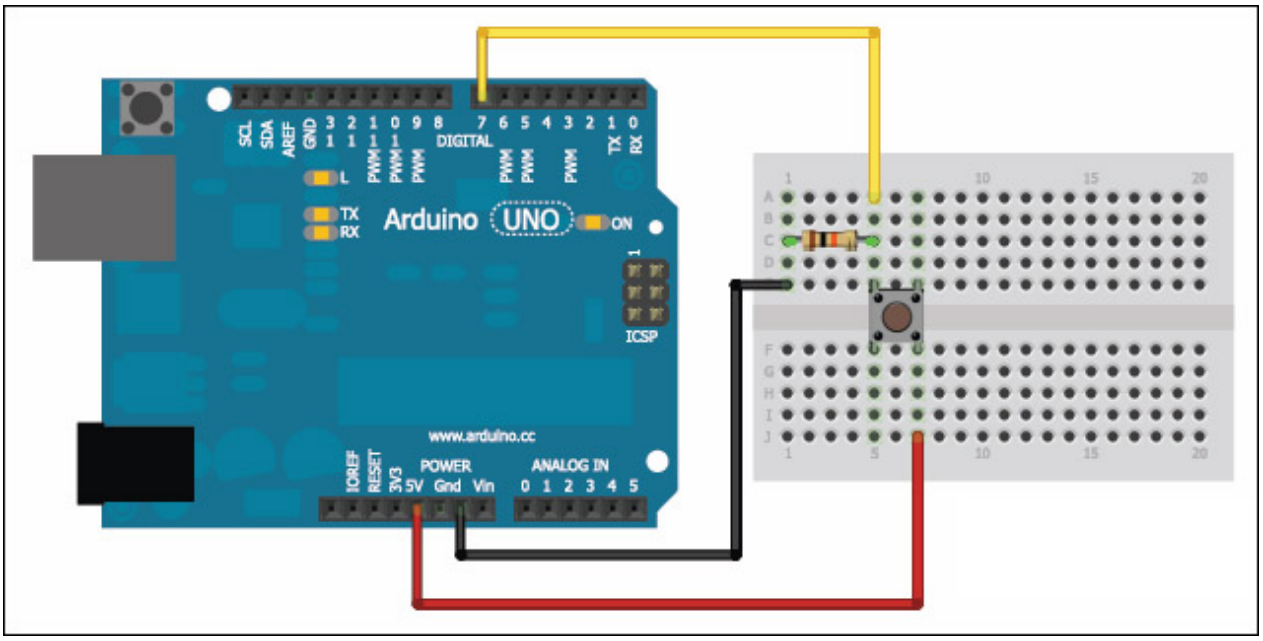
Se il pulsante non è premuto, dobbiamo cominciare a scorrere i due array per leggere le varie note e suonarle. Possiamo farlo con un ciclo **for**: i cicli for hanno una variabile contatore: nel nostro caso la variabile **i**. Il valore iniziale della variabile **i** è **0**, e ad ogni ciclo il suo valore 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**.