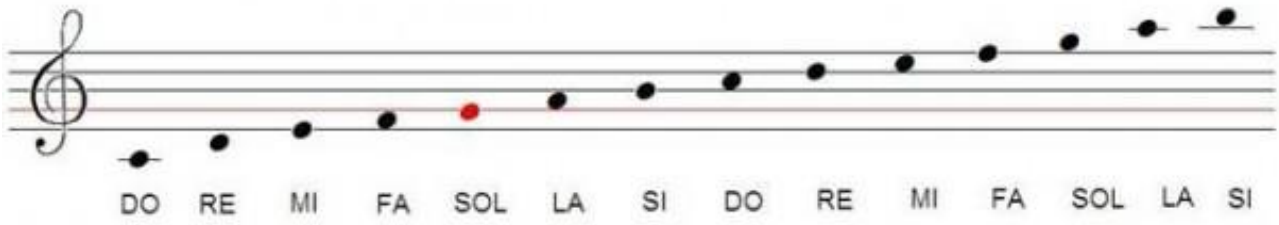
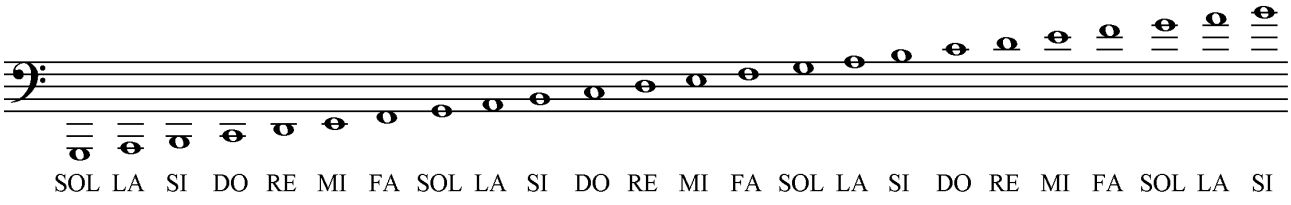
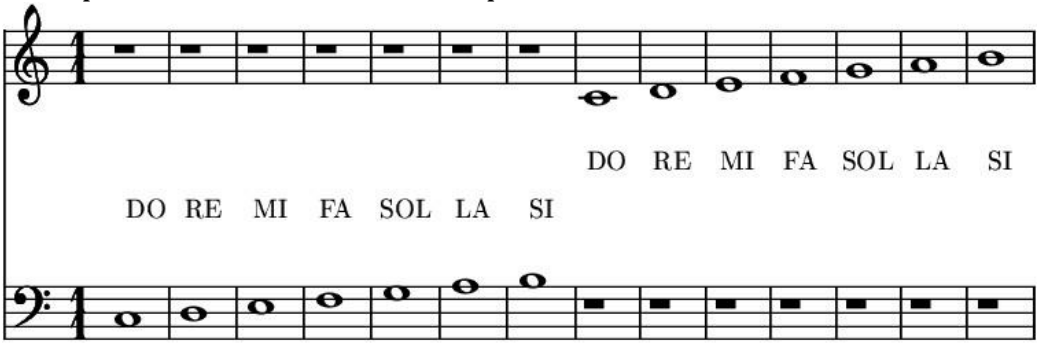
Anni e anni spesi a imparare teoria musicale… mesi e mesi, ore e ore impiegate a esercitarsi sullo strumento… quanto sarebbe bello se un robot suonasse per noi la nostra melodia preferita…
Ehi aspetta c'è Arduino!!!
Zumo 32u4 è un piccolo robot programmabile in C e C++. Molto simile al comune Arduino, l'azienda creatrice Pololu ha integrato, grazie a particolari librerie, la possibilità di "insegnare" al robottino a suonare.
Vediamo nel dettaglio come fare.
Iniziamo con un po' di teoria musicale.
Spesso di fronte ad uno spartito non sappiamo cosa e dove guardare… impariamo prima a leggere la musica, e poi vedremo come farla riprodurre al robot.



Possiamo vedere in figura le note scritte sul pentagramma. Dal Do in basso a sinistra fino al Si di un'ottava superiore. Le note sono sette: do, re, mi, fa, sol, la, si. Dopo il Si si ricomincia dal do: questo do però sarà un'ottava superiore rispetto a quello iniziale, cioè vuol dire che sarà otto posizioni più avanti nel pentagramma. In gergo musicale si dice che sarà un'ottava superiore. In poche parole il do che si trova al centro del pentagramma è più acuto di quello in basso a sinistra.
All'inizio del pentagramma vediamo uno strano simbolo. È la famosa chiave di violino, o di Sol. In questo caso non ci serve a molto però è importate sapere che ci sono differenze tra le diverse chiavi che esistono. Ad esempio uno spartito per pianoforte si compone di due linee di pentagramma: una chiave di violino (note acute che verranno riprodotte dalla mano destra), e una chiave di basso (note più gravi che saranno suonate dalla mano sinistra). La principale differenza tra la chiave di violino e quella di basso è che, se messe a confronto, la lettura delle note non corrisponde.



Come si può facilmente vedere le note scritte in chiave di basso non coincidono con quelle scritte in chiave di violino, ma sono spostate di due posizioni sopra: il do della prima figura corrisponde al mi se messo in uno spartito con chiave di basso.



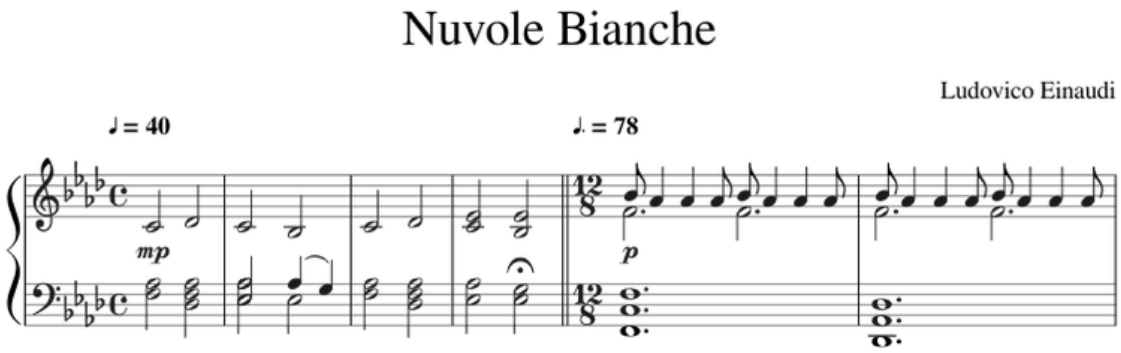
Questo è un esempio di spartito per pianoforte. È impossibile riprodurre questa sequenza con Arduino. Essendo solo un piccolo robottino datato di buzzer, Zumo è in grado di riprodurre solo una serie di note, non di leggere uno spartito. Per suonare una serie del genere servirebbero almeno due robottini, oppure più di un buzzer.

Questo è il primo passo. Il passo successivo è imparare la notazione inglese.
Essa è:

Do	*Re*	*Mi*	*Fa*	*Sol*	*La*	*Si*	*Do*
C	D	E	F	G	A	B	C

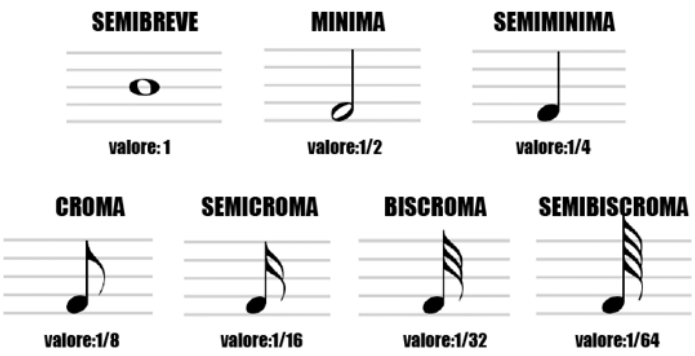
Adesso insegniamo al robot a riprodurre qualcosa.
Il programma è costruito in modo tale che la prima informazione da immettere sia il tempo. In musica canzoni diverse hanno ritmi e tempi diversi. Ritmo e tempo non sono la stessa cosa, però per intenderci ci sono canzoni che sono più veloci di altre. Un brano di musica classica non avrà lo stesso tempo di un pezzo di musica leggera. In musica si parla di BPM (battiti per minuto).
Ecco un esempio:

La nota indica esattamente il tempo del brano, che in questo caso corrisponde a 65bpm.
Il tempo può anche variare durante un brano, come in questo caso:



In questo caso si parte con 40bpm, per poi arrivare a 78bpm. Nel codice il tempo è la prima cosa da scrivere e si indica così: "!T65"(ovviamente al posto di 65 va inserito il corretto tempo della canzone).
In questo esempio, vicino la chiave di sol e di basso ci sono dei simboli. Quelle piccole b significano bemolle, cioè le note la, si, re e mi devono essere suonate diminuite di un semitono, cioè avremo il la bemolle, il si bemolle e così via. Perché proprio il la, il si, il re e il mi? I quattro simboli di bemolle si trovano sullo spartito proprio in corrispondenza di quelle note (si veda la prima figura per conferma). Al posto del simbolo di bemolle si sarebbe potuto essere anche questo simbolo "#", che vuol dire diesis: la nota invece di essere diminuita di un semitono è invece, in questo caso aumentata di un semitono. Dire la # o si b è la stessa cosa (non è proprio la stessa cosa, però in questo caso non fa differenza… per un vero musicista c'è differenza).
Arduino inoltre non può suonare due note contemporaneamente. Nella figura sovrastante dopo il simbolo 12/8 ci sono due note che si trovano esattamente una sopra l'altra. In questo caso il consiglio è scegliere di scrivere nel codice solo quella più acuta, perché è quella della melodia.

Dopo il tempo bisogna indicare la durata delle note che si vogliono riprodurre.



Ci sono diversi valori di note. Nel codice bisogna scrivere L1 se il valore della nota o delle note che si vogliono suonare sono semibrevi, L2 se sono minime, L4 se sono semiminime, L8, L16, L32 e L64 rispettivamente per crome, semicrome, biscrome e semibiscrome.

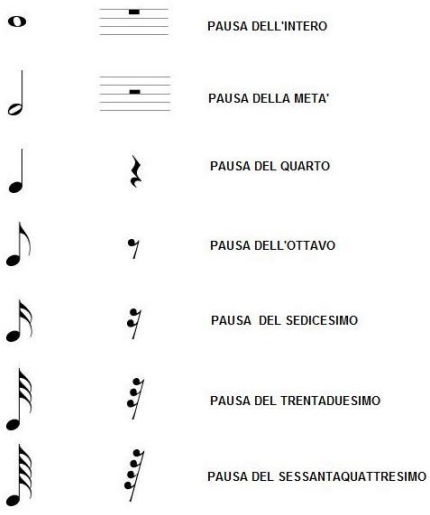
Il tempo e la lunghezza delle note si possono variare in qualsiasi momento. Ecco un esempio di codice e di variazione di questo tipo.

```
//playing = true;
buzzer.play("!T92 L16 ebabgbf+be.R !T105 L8 a-gfa-gf>cgfa-gfa-gfa-gf>d-ga-a-gf >cb-a->cb-a->e-a-b->cb-a-b-a-g>e-ga-b-a-g
```

buzzer.play(“”); è il nome della funzione. All’interno delle parentesi tonde, tra le virgolette va indicato per primo il tempo (!T92), poi la lunghezza delle note, in questo caso semicrome, e infine si può iniziare con la scrittura vera e propria delle note. I simboli > e < prima di una nota indicano che essa è un’ottava superiore. Qui in basso vi è una leggenda con tutti i simboli, utilissima per le capire che le R corrispondono alle pause, i “+” ai diesis, i “- ” ai bemolle.

Control character(s)	Effect
A–G	Specifies a note that will be played.
R	Specifies a rest (no sound for the duration of the note).
+ or **#** after a note	Raises the preceding note one half-step.
- after a note	Lowers the preceding note one half-step.
1–2000 after a note	Determines the duration of the preceding note. For example, C16 specifies C played as a sixteenth note (1/16th the length of a whole note).
. after a note	"Dots" the preceding note, increasing the length by 50%. Each additional dot adds half as much as the previous dot, so that "A.." is 1.75 times the length of "A".
> before a note	Plays the following note one octave higher.
< before a note	Plays the following note one octave lower.
O followed by a number	Sets the octave. (default: **O4**)
T followed by a number	Sets the tempo in beats per minute (BPM). (default: **T120**)
L followed by a number	Sets the default note duration to the type specified by the number: 4 for quarter notes, 8 for eighth notes, 16 for sixteenth notes, etc. (default: **L4**)
V followed by a number	Sets the music volume (0–15). (default: **V15**)
MS	Sets all subsequent notes to play play staccato – each note is played for 1/2 of its allotted time, followed by an equal period of silence.
ML	Sets all subsequent notes to play legato – each note is played for full length. This is the default setting.
!	Resets the octave, tempo, duration, volume, and staccato setting to their default values. These settings persist from one `play()` to the next, which allows you to more conveniently break up your music into reusable sections.

Le pause (“R”) hanno valori diversi in base alla lunghezza delle note (L) indicata: “L8 R” vuol dire che c’è una pausa di un ottavo.
Ecco qui illustrate le diverse pause che si possono trovare su uno spartito.



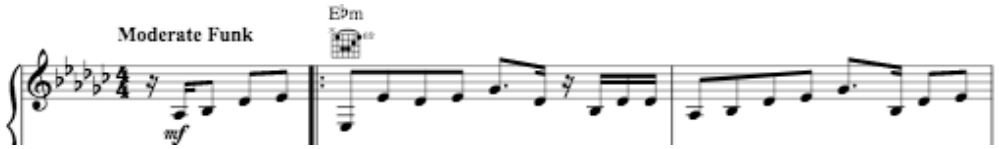
È importantissimo mettere dopo la funzione buzzer.play un delay abbastanza lungo da coprire l’intera riproduzione del brano, oppure una funzione il cui tempo di azione sia però abbastanza lungo da non interrompere la canzone durante la sua esecuzione.

Ecco spiegato tutto. Qui di seguito alcuni esempi nel caso in cui qualcosa non fosse chiaro.

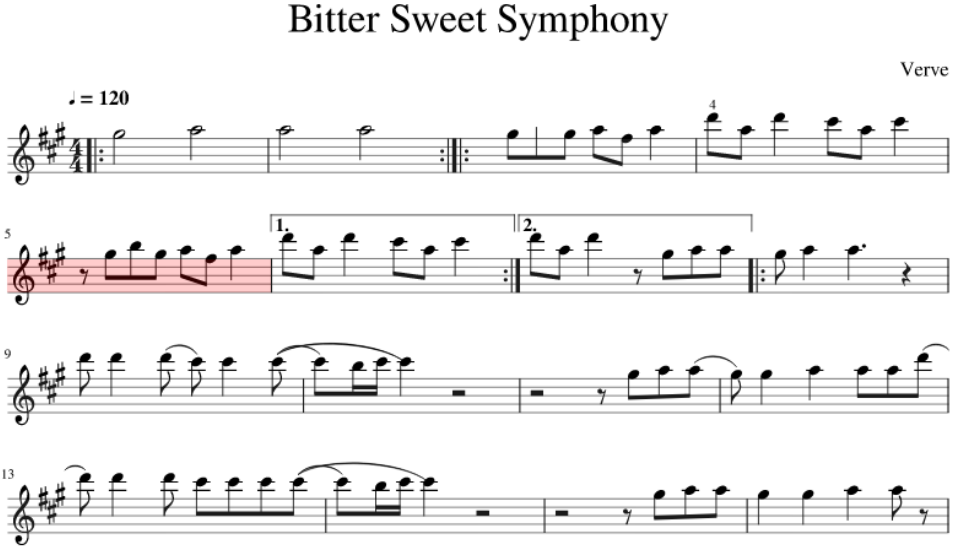
BUON DIVERTIMENTO!!!!

```
buzzer.play("!T108 L16 >g-RRd-RR- L8 e-<e-e-d-e->g- L16Rd-RR L8e-<a-<b-d-e->g-L16 Rd-RR L8 e-<e-e-d-e->g-L1
delay(500000);
```

Superstition – Stevie Wonder.



Bitter Sweet Symphony – The Verve




```
buzzer.play("!T90 L8 Rg+bg+af+aa>da>d>d>c+a>c+>c+Rg+bg+af+aa>da>d>c+a>c+>c+g+bg+af+aabab>c+a>c
delay(500000);
  buzzer.stopPlaying();
```

RICCARDO TANCREDI V^C