TP: música con Arduino

Aquí continuaremos trabajando con la plataforma Arduino, en particular veremos una aplicación puntual de las salidas PWM y los buzzer pasivos.

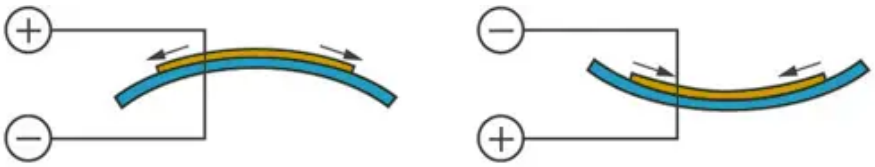
¿Qué es un buzzer?

Un buzzer es un dispositivo que permite convertir una señal eléctrica en una onda de sonido. Existen de dos tipos:

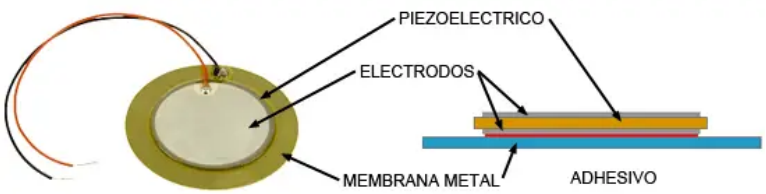
- **Buzzer activo**: los buzzer activos disponen de un oscilador interno, por lo que únicamente tenemos que alimentar el dispositivo para que se produzca el sonido en un tono fijo.
- **Buzzer pasivo o altavoz**: dispositivos no disponen de electrónica interna, por lo que tenemos que proporcionar una señal eléctrica variante en el tiempo para conseguir el sonido deseado, nos permiten de esta manera generar diversos tonos (¡ya los usamos!).

¿Como funcionan? (repaso)

Técnicamente los buzzers son transductores electroacústicos, es decir, dispositivos que convierten señales eléctricas en sonido. Los buzzer son transductores piezoeléctricos. Los materiales piezoeléctricos tienen la propiedad especial de variar su volumen al ser atravesados por corrientes eléctricas.



Un buzzer aprovecha este fenómeno para hacer vibrar una membrana al atravesar el material piezoeléctrico con una señal eléctrica. En el caso del buzzer pasivo la señal que aplicamos (con una pequeña etapa de potencia) hacer vibrar la membrana en la frecuencia aplicada, mientras que en el caso del buzzer activo el oscilador aplica siempre la misma frecuencia.



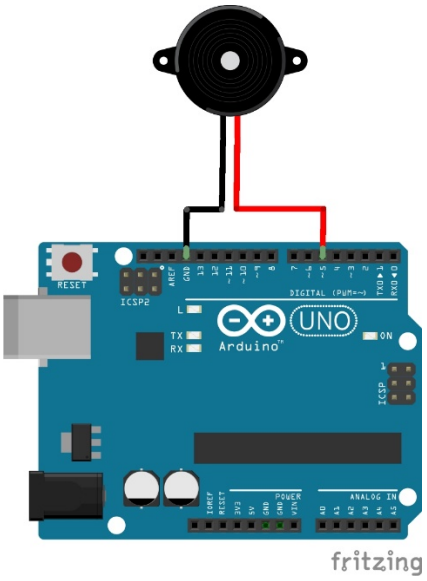
¿Cómo diferenciar un buzzer activo de uno pasivo?

Los buzzers activos producen un tono audible fijo, con solo aplicar una tensión continua (usualmente 3.3V a 5V) generara un tono, Mientras que si alimentamos con una señal continua a un buzzer pasivo este no genera ningún ruido (requiere una señal oscilante, en general de tipo PWM, que indique la frecuencia y la duración de la señal).

Encapsulado buzzer (o zumbador), notar que es un dispositivo con polaridad, al momento de conectarlo debemos respetar su pata positiva indicada por el símbolo +

Conexión buzzer activo:

En el caso del buzzer activo solo debemos conectar a su pata positiva una pata digital de nuestro Arduino, y una conexión a GND en la otra pata del buzzer. Para lograr un sonido solo debemos activar la pata digital. A continuación, un ejemplo:



Nota: La pata positiva del buzzer es la que debe conectarse al pin digital de Arduino.

Programa:

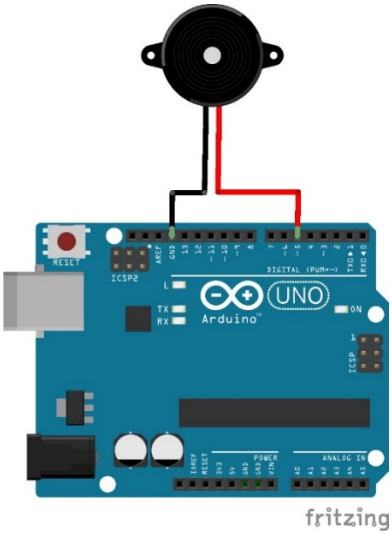
```
// Ejemplo: led que parpadea
void setup()
{
 pinMode(5, OUTPUT);
}
void loop()
{
 digitalWrite(5, HIGH);   // enciende el buzzer
 delay(1000);             // espera por un segundo
 digitalWrite(5, LOW);    // apaga el buzzer
 delay(1000);             // espera por un segundo
}
```

Explicación: el programa encenderá y apagara el buzzer por un segundo, como en este caso es un buzzer activo tendremos un tono fijo cada vez que se encienda.

Conexión buzzer pasivo:

En el caso del buzzer pasivo debemos conectar a su pata positiva una pata capaz de generar una señal variante, para el caso del Arduino esta serán los pines con capacidad de PWM. A continuación, dos ejemplos:

Ejemplo 1:



Nota: La pata positiva del buzzer es la que debe conectarse al pin PWM de Arduino.

```
// Ejemplo: buzzer simple
#define buzzer 5 // pin buzzer
#define pausa 100 // pausa del beep
int duty_cycle = 40; // valor del PWM
void setup()
{
}
void loop()
{
analogWrite(buzzer, duty_cycle);   //Encendido del buzzer a un tono determinado
delay(pausa);              // Espera
analogWrite(buzzer, 0);          // Apaga
delay(pausa);
}
```

Explicación: el programa encenderá y apagará el buzzer por el tiempo definido en pausa (será la duración de nuestro tono), el valor de PWM definido en la variable duty_cycle definirá la frecuencia del tono a escuchar.

Funcion tone()

Arduino dispone de dos funciones que nos permiten generar fácilmente señales eléctricas para convertir en sonido, usando cualquiera de las salidas digitales disponibles. Estas funciones son tone() y noTone() y, como su nombre indican, permiten generar o detener la señal del tono en un pin.

tone(pin, frequencia, duracion);

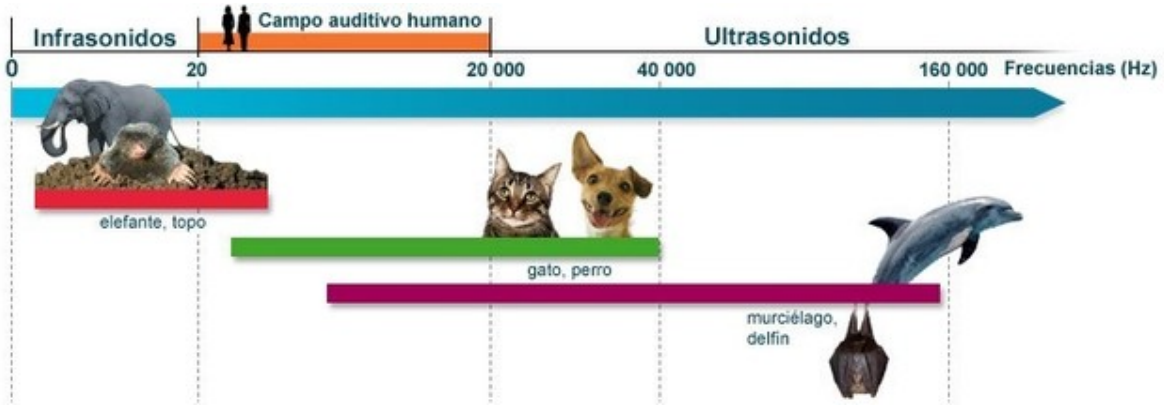
Esta función activa un tono de frecuencia (31 Hz a 65535 Hz) y duración (expresada en ms) determinados en un pin dado. El parámetro duración es opcional.

noTone(pin);

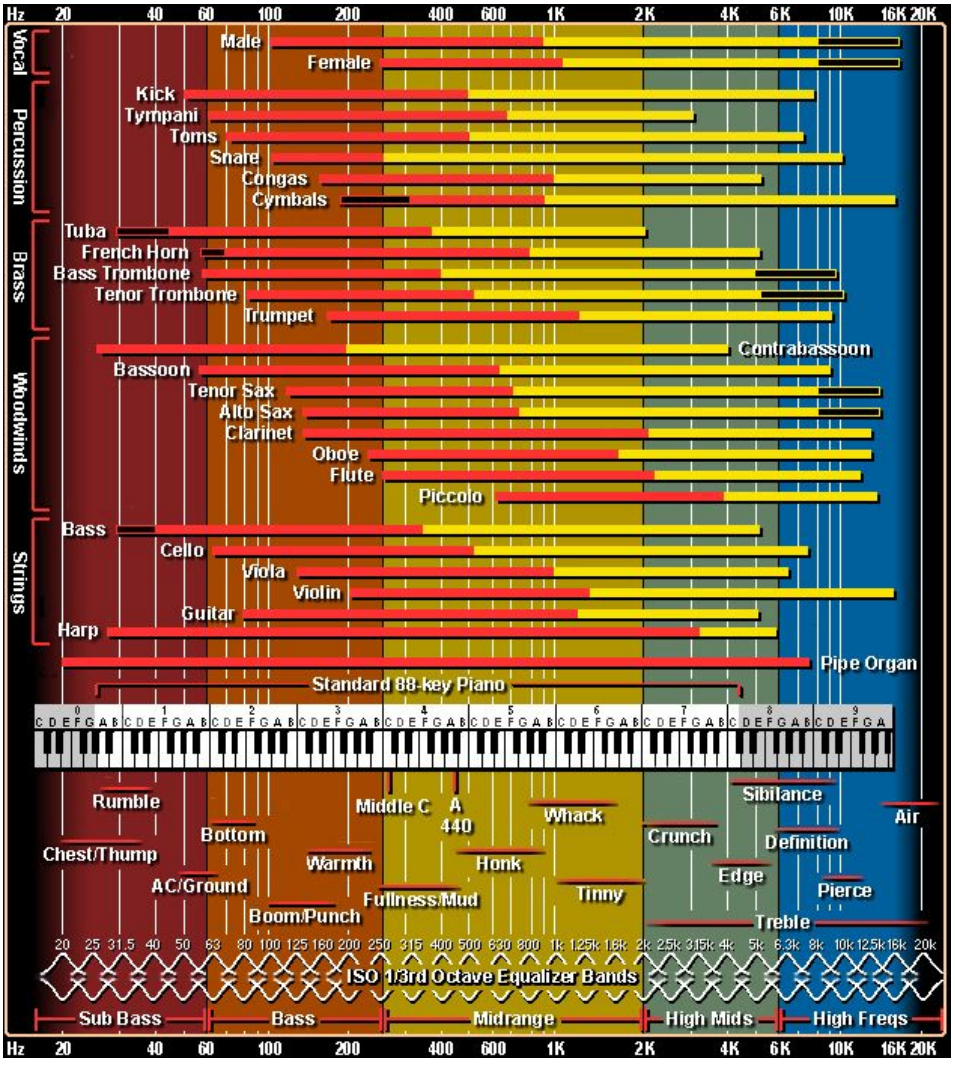
Esta función detiene el tono en ejecución en el pin dado.

A al usar las funciones para la generación de tone tenemos que asumir importantes limitaciones.

- La función Tone emplea el Timer 2, por lo que mientras esté funcionando no podremos usar las salidas PWM en los pines 3 y 11 en Arduino Nano y Uno (pines 9 y 10 en Arduino Mega).
- No podemos usar la función tone() en dos pines de forma simultánea. Deberemos apagar el tono con la funcion noTone() antes de poder usarlo en otro pin.
- Los rangos de la función tone son de 31 Hz a 65535 Hz.



Detalle de distintos rangos auditivos



Detalle de distintos rangos de frecuencia para distintos instrumentos

Ejemplo 2:

El siguiente código muestra el uso de estas funciones en un ejemplo simple, en el que empleamos el buzzer conectado en el Pin 5 (como en el diagrama anterior) para generar una función de 440Hz durante un segundo, pararlo durante 500ms, y finalmente un tono de 523Hz durante 300ms, para repetir el programa tras una pausa de 500ms.

```
#define pinBuzzer  5
void setup()
{
}
void loop()
 {
//generar tono de 440Hz durante 1000 ms
tone(pinBuzzer, 440);
delay(1000);
//detener tono durante 500ms
noTone(pinBuzzer);
delay(500);
//generar tono de 523Hz durante 300ms, y detenerlo durante 500ms.
tone(pinBuzzer, 523, 300);
delay(500);
}
```

Concepto de tabla (o array, o matriz) en programación

Antes de continuar algo que nos será útil de conocer es el concepto de tabla o array en programación. Básicamente se trata de un conjunto de variables del mismo tipo (int, float, long, etc) todas juntas en la memoria y referenciadas mediante un único nombre y un numero llamado índice. Este elemento es útil cuando tenemos mucha información con la que trabajar, en lugar de utilizar una variable para cada elemento (imagínense si tuviésemos 100 o 1000 elementos a trabajar) solo utilizamos un nombre y un numero en el rango de elementos a trabajar.

Para trabajar con tablas debemos **declararlas**, al igual que las variables. La forma de declaración es el tipo de dato, el nombre, y el tamaño encerrado entre corchetes. Veamos distintos ejemplos:

Código	Explicación
int myInts[6];	Define una tabla de 6 elementos tipo int, y no se dan valores a los mismos.
int myPins[] = {2, 4, 8, 3, 6};	Define una tabla de tipo int con los números 2 4 8 3 6, ver que en este caso no se impuso el tamaño entre corchetes, sino que el compilador automáticamente lo definirá de los números a guardar en ella.
int mySensVals[6] = {2, 4, -8, 3, 2};	Define una tabla de 6 elementos tipo int, donde se asignan los primeros cinco con valores.
int tabla[5][10];	Define una tabla de tipo int de 5 filas por 10 columna, en total podremos guardar en ella 50 elementos de tipo int.

Para trabajar con tablas, lo realizaremos con cada elemento como si fuese una variable independiente. Para acceder a cada elemento utilizaremos el nombre de la tabla y entre corchetes indicaremos el índice de acceso. **El índice de la tabla siempre inicia en cero y llegara hasta el largo de la misma menos 1.**

Por ejemplo, si tenemos la siguiente definición:

int tabla[6] = {2, 4, -8, 3, 2};

Si escribimos:

A = tabla[0];

En la variable A tendremos el valor 2 (guardado en la primera posición de tabla.

En forma similar si queremos guardar algo en una tabla podremos escribir su nombre junto a su índice, por ejemplo:

tabla[2]=25;

Siguiendo el ejemplo la línea anterior guardará el valor 25 en la posición 2 de la tabla (se escribirá sobre el valor -8 que estaba previamente).

Atención con los índices, el IDE no nos informa si utilizamos un índice por fuera del rango correcto para nuestra tabla.

Ejemplo 3:

Mezclemos un poco de tablas y el uso del buzzer, con el mismo circuito presentado previamente. El siguiente programa utiliza una tabla con valores de frecuencias, cada uno de los elementos de la tabla los recorremos secuencialmente utilizando un bucle for. Como resultado obtendremos un barrido que se aproxima las distintas notas musicales.

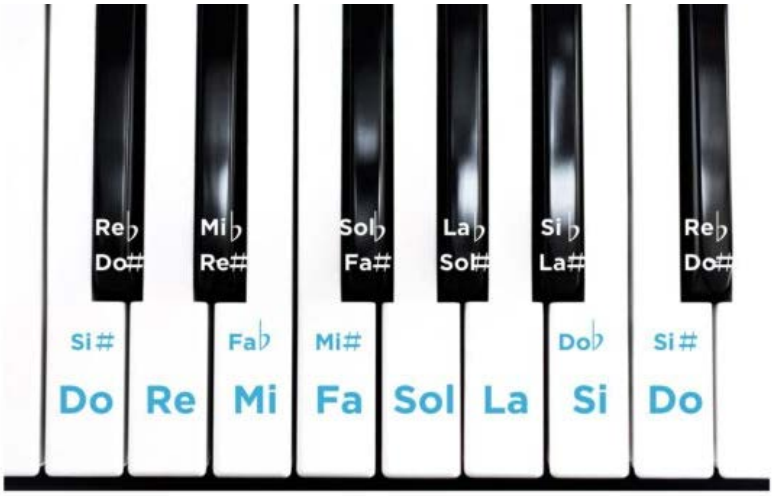
```
#define pinBuzzer  5
// tabla con distintos valores de frecuencias
int tonos[] = {261, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494};
int i =0;
void setup()
{
 }
void loop()
{
        for (i = 0; i < 10; i++)
        {
                tone(pinBuzzer, tonos[i]);
                delay(1000);
        }
        noTone(pinBuzzer);
}
```

Un poco de teoría de musical

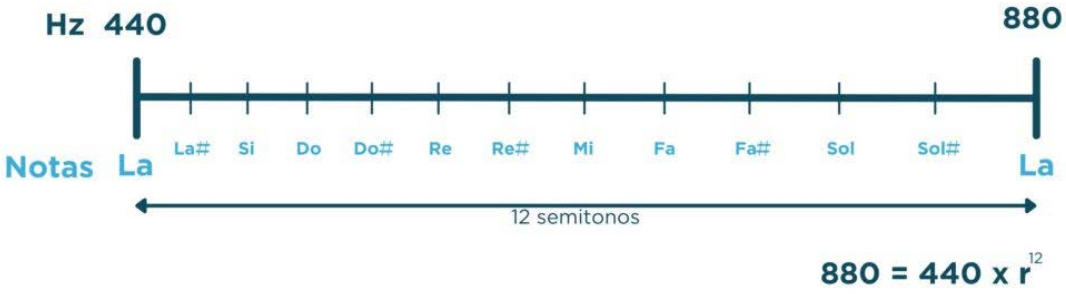
Al poder reproducir tonos por un cierto tiempo, el siguiente desafío seria intentar reproducir alguna canción o melodía que nos interese. Para esto deberíamos tener que interpretar un poco de teoría musical (no mucho realmente 😊).

Conocemos que una nota musical en principio es un tono a una frecuencia determinada. Las notas se arreglan dividiendo una octava (relación 2:1 entre un tono) en 12 semitonos. Cada uno de estos se encuentran a una frecuencia dada, según la octava que se quiera, vea la tabla al fin de la clase.

Nota: Una octava es un arreglo de tonos, permitiendo tener notas más graves (mayor octava) o agudas (menor octava) asociadas a los mismos tonos.



Semitonos de una octava en el teclado de un piano



Ejemplo de una octava entre dos notas La, vemos que existen 12 semitonos, el # indica el sostenido, es medio tono por sobre la nota.

Por ejemplo, si tomamos la 4ta octava las notas tienen las siguientes frecuencias:

- Do1 = 261.63
- Re = 293.66
- Mi = 329.63
- Fa = 349.23
- Sol = 392.00
- La = 440.00
- Si = 493.88
- Do2 = 523.63 (ya es de la 5ta octava)

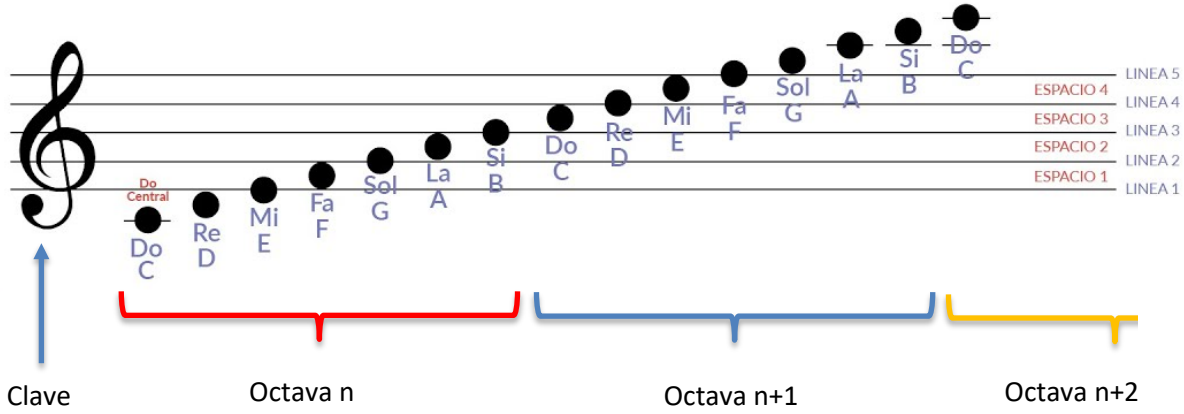
Dado que la diferencia al oído entre, por ejemplo, 261.63 Hz y 261 Hz no es apreciable, para poder hacer un uso más eficiente de la memoria de nuestro Arduino vamos a redondear estas cifras para que sean más manejables.

- Do1 = 261
- Re = 294
- Mi = 330
- Fa = 349
- Sol = 392
- La = 440
- Si = 494
- Do2 = 524
-

Asignaremos estas frecuencias a unas **constantes**. Si los números que manejamos son con coma deberemos utilizar float y si son sin comas podemos utilizar int. En el primer caso la placa reserva más memoria para estos datos que en el segundo. En nuestro caso, dado que la diferencia de sonido es inapreciable, redondearemos para no utilizar comas y utilizaremos int.

¿Como leer una partitura?

Si bien es un tema complejo, aquí daremos algunas indicaciones básicas, iniciaremos con conocer cómo se escriben las distintas notas en un pentagrama, en particular en la clave de Sol (símbolo al inicio del pentagrama que indica donde está la nota sol en el mismo) por ser de las más comunes:

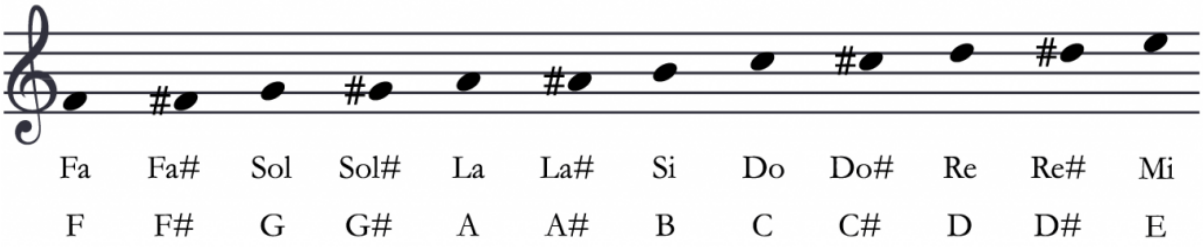


En la imagen podemos ver un pentagrama donde se indican cada una de las notas en su lugar, conocemos que las notas van del Do al Si, pero si duplicamos la frecuencia del Do obtendremos nuevamente un Do pero una octava más arriba, y si la dividimos por dos obtendremos una octava más abajo.

Observando el pentagrama podremos definir que nota corresponde, luego eligiendo la octava a la que deseemos tocar terminaremos de definir que frecuencias serán estas notas. Como se observa en el pentagrama anterior es posible mezclar notas de distintas octavas por ejemplo tendremos un Do en la octava 4 y podríamos tener un Do en la octava 5.

Para nuestro caso trabajaremos en la octava 4 que es una de las más utilizadas, y nos da un poco de espacio para poder aprovechar octavas por arriba y por debajo de la misma.

La siguiente imagen muestra un pentagrama con el agregado de ver las posiciones de los semitonos entre las notas antes presentadas (los sostenidos se les dice # y a los bemoles se les dice ♭)



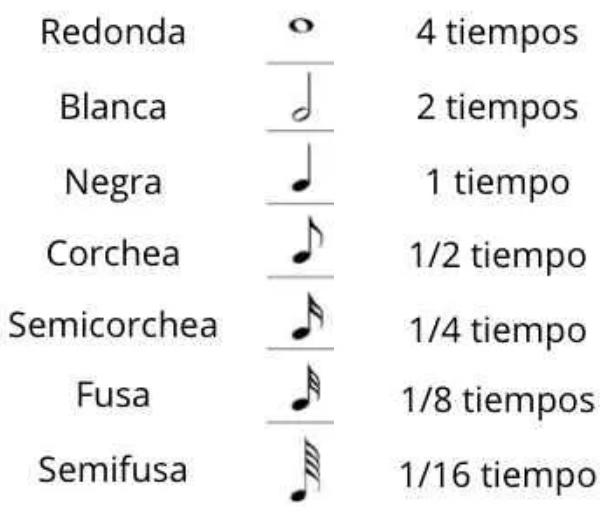
- El sostenido (#) eleva el tono de una nota un semitono cromático.
- El bemol (♭) baja el tono de una nota un semitono cromático.

Como vimos recién la **posición vertical** indica la nota, notar que en la imagen anterior se dan las dos nomenclaturas para las notas: Do, Re, Mi, … que es la versión de notación musical latina y la versión anglosajona basada en letras mayúsculas.

La **posición horizontal** de la nota define cuándo es emitida. Así, el eje horizontal del pentagrama define una escala de tiempo creciente desde la izquierda hacia la derecha.

La **forma** de la nota (como la dibujamos) define su **duración**. Las duraciones estándar de notas están definidas en solfeo, donde cada una dura la mitad que la anterior.

Nota: a veces por una cuestión de escritura “los palos” de las notas van hacia arriba o hacia abajo, esto no cambia su duración. Se toma la dirección en que el dibujo quede más centrado en el pentagrama. De la misma forma cuando dos formas similares son consecutivas se las une gráficamente, esto tampoco cambia el sonido o la duración de estas.

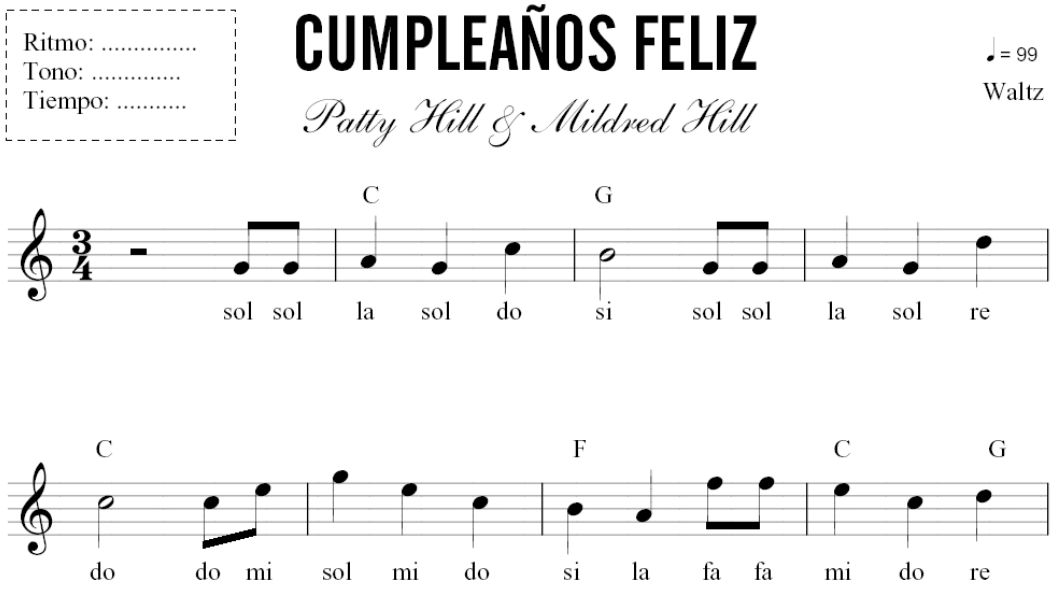


El tiempo es la duración de la nota. Cada figura tendrá un tiempo determinado por el tiempo de la negra (♩), de tal manera que una blanca durará doble que una negra y una corchea la mitad. La relación de los tiempos de las figuras es fija, pero el tempo lo vamos a marcar nosotros (en realidad cada partitura lo indicará, pero para nuestro caso será un valor de tiempo constante).

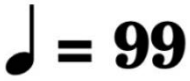
Por ejemplo, si tomamos un segundo para una negra tendremos:

Figura	Tiempo(ms)
Redonda	4000
Blanca	2000
Negra	1000
Corchea	500
Semicorchea	250
Fusa	125
Semifusa	62

Ejemplo: Cumpleaños Feliz



¿Como iniciamos? Lo primero es definir el valor de tempo (tiempo de negra), en esta partitura lo tenemos indicado como:



En este caso el tempo es 99, esto quiere decir que en 1 minuto (60 segundos) entran 99 negras. Con una sencilla regla de tres podemos deducir que una negra dura 600 milisegundos.

Así pues, damos un valor de 600 ms para la negra, por lo tanto:

- Negra 600ms
- Blanca (negrax2) 1200ms
- Corchea (negra/2) 300ms
-

Si nos ponemos a programar esta melodía, a pesar de ser sencilla y corta, comprobaremos que es muy laborioso ir consultando la frecuencia de cada nota para incluirla en la función Tone. Una forma más fácil es armar una tabla a partir tanto de las notas como de los tiempos de estas, veamos cómo queda (tiene el programa para cargar y probar directamente, usando el mismo armado mostrado anteriormente):


```
#define buzzer 5
//Definicion de notas
const int Do = 261;   //octava 4
const int Re = 294;
const int Mi = 330;
const int Fa = 349;
const int Sol = 392;
const int La = 440;
const int Si = 494;
const int Do2 = 524;  //octava 5
const int Re2 = 588;
const int Mi2 = 660;
const int Fa2 = 699;
const int Sol2 = 785;

//Definicion de tiempos
const int n = 600;  //Duración en ms de las notas negras
const int b = n*2;  //Duración en ms de las notas blancas, relativo al valor de una negra
const int c = n/2;  //Duración en ms de las notas  corchea, relativo al valor de una negra

//Cancion en forma de tabla
const int cancion[24][2] =
{
{Sol, c}, {Sol, c}, {La, n}, {Sol, n}, {Do2, n}, {Si, b}, {Sol, c}, {Sol, c}, {La, n}, {Sol, n}, {Re, n}, {Do2, b},
{Do2, n}, {Mi2, n}, {Sol2, n}, {Mi2, n}, {Do2, n}, {Si, n}, {La, n}, {Fa2, n},  {Fa2, n}, {Mi2, n}, {Do2, n},
{Re2, n},
};
//Programa
void setup() {
}
void loop() {
//Bucle que recorre 24 elementos de la tabla
for (int i = 0; i < 23; i++)
{
tone(buzzer, cancion[i][0]); //aplico frecuencia del elemento i de la tabla
delay(cancion[i][1]); //aplico duracion del elemento i de la tabla
noTone(buzzer); //apago tono para esta listo para el proximo
}
delay(1000); //retardo entre repeticiones de la cancion
}
```

Explicación: Vemos primero la definición de las notas, en nuestro caso de la 4ta y 5ta octavas. Luego encontramos la definición de tiempos como n (negra) de 600, b (blanca) y c (corchea).

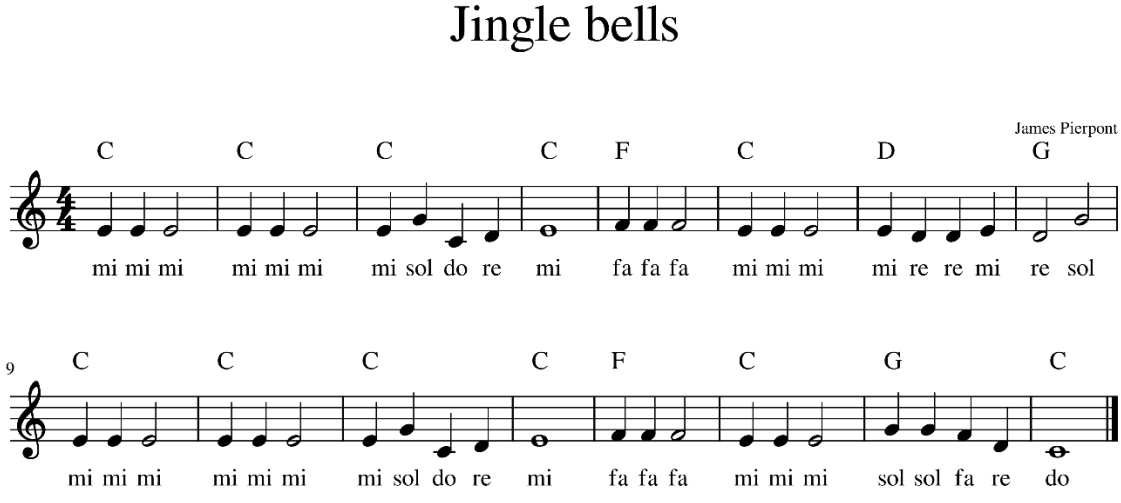
Luego encontramos una tabla de 24 filas y dos columnas, cada fila contendrá en la primera columna la nota y en la segunda su duración. Ver que es la traducción del pentagrama a un formato que podremos ejecutar con nuestro Arduino.

Finalmente, en el loop de nuestro programa encontramos un bucle for que se realizara 24 veces (coincidiendo con el número de notas de nuestra canción), en cada vuelta del for, se utiliza la función tone accediendo al elemento de la nota de la tabla (recordar que entre corchetes van los índices de la tabla, el primero accede al número de fila y el segundo al número de columna), y un delay para dar el tiempo a cada nota (donde el tiempo del delay se accede también desde la tabla).

Finalmente encontramos un notone(); para que finalice el sonido entre tonos.

Actividad

1) **A partir de la siguiente partitura escriba la canción en el formato que Arduino pueda reproducirla, tome como tempo 129:**



2) **Pruebe los programa adjuntos indicados como A,B,C y trate de identificar que canciones son.**

3) **Buscar y convertir una música que quieran:**

La tarea de convertir al formato de tonos una música cualquiera (como las del punto 2), siguiendo el método desarrollado en 1) para canciones simples puede funcionar, pero en canciones más largas se hace algo tedioso. Podemos aprovechar algunas aplicaciones libres que hacen esto por nosotros.

A) **Iniciaremos por buscar un archivo Midi de la canción que nos interese, en internet hay varios sitios que los proporcionan, aquí te dejamos algunos para que busques:**
   http://www.musicamidigratis.com/
   midis.com.ar

B) **Aunque no es necesario, es bueno comprobar que el archivo Midi descargado sea el buscado, para esto podemos utilizar cualquier reproductor de archivos Midi, a continuación, le dejamos el enlace a uno online (solo arrastre el archivo descargado sobre la pantalla):**
   https://midi.fullpartituras.com/

C) **Utilice un converso de formato Midi al formato de tabla compatible con Arduino. A continuación, le dejamos uno que directamente obtiene como salida el archivo con formato de proyecto Arduino (.INO) para pasar a la placa (corrija el lugar donde está conectado su buzzer).**
   https://arduinomidi.netlify.app/

   Este conversor da una tabla similar a la que utilizamos, pero agrega una tercera columna que representa un tiempo de silencio entre tonos.

¡Atención con el tamaño de las canciones! La memoria de nuestra placa Arduino es finita, por lo que, si buscamos una canción muy larga, al momento de descargarla en la placa nos indique hemos usado toda la memoria disponible. En estos casos tendremos que recortar a la mano la canción eliminando partes de la tabla que la compone, recuerde que la tabla además del contenido tiene su definición donde se indica el tamaño de esta.

¿Qué es un archivo MIDI? Un archivo MIDI es un archivo de interfaz digital de un instrumento musical. No contienen datos de audio propiamente dichos, por lo que son mucho más pequeños. Contiene una serie de instrucciones que sólo puede entender un instrumento compatible con MIDI. Actúa como instrucciones que explican cómo debe producirse el sonido. Explica qué notas se tocan, cuándo se tocan y durante cuánto tiempo debe sonar cada nota.

Cuando usamos el conversor MIDI – Arduino veremos que nos da la posibilidad de distintos archivos de salida, dependiendo del archivo MIDI original puede que nos devuelva el sonido de cada instrumento por separado.

Frecuencias de notas musicales

Nota	Octava	Frecuencia (Hz)
Do / C	0	16,35
Do# / C#	0	17,32
Re / D	0	18,35
Re# / D#	0	19,45
Mi / E	0	20,60
Fa / F	0	21,83
Fa# / F#	0	23,12
Sol / G	0	24,50
Sol# / G#	0	25,96
La / A	0	27,50
La# / A#	0	29,14
Si / B	0	30,87
Do / C	1	32,70
Do# / C#	1	34,65
Re / D	1	36,71
Re# / D#	1	38,89
Mi / E	1	41,20
Fa / F	1	43,65
Fa# / F#	1	46,25
Sol / G	1	49,00

Sol# / G#	1	51,91
La / A	1	55,00
La# / A#	1	58,27
Si / B	1	61,74
Do / C	2	65,41
Do# / C#	2	69,30
Re / D	2	73,42
Re# / D#	2	77,78
Mi / E	2	82,41
Fa / F	2	87,31
Fa# / F#	2	92,50
Sol / G	2	98,00
Sol# / G#	2	103,83
La / A	2	110,00
La# / A#	2	116,54
Si / B	2	123,47
Do / C	3	130,81
Do# / C#	3	138,59
Re / D	3	146,83
Re# / D#	3	155,56
Mi / E	3	164,81
Fa / F	3	174,61
Fa# / F#	3	185,00
Sol / G	3	196,00
Sol# / G#	3	207,65
La / A	3	220,00
La# / A#	3	233,08
Si / B	3	246,94
Do / C	4	261,63
Do# / C#	4	277,18
Re / D	4	293,67
Re# / D#	4	311,13
Mi / E	4	329,63
Fa / F	4	349,23

Fa# / F#	4	369,99
Sol / G	4	392,00
Sol# / G#	4	415,31
La / A	4	440,00
La# / A#	4	466,16
Si / B	4	493,88
Do / C	5	523,25
Do# / C#	5	554,37
Re / D	5	587,33
Re# / D#	5	622,25
Mi / E	5	659,26
Fa / F	5	698,46
Fa# / F#	5	739,99
Sol / G	5	783,99
Sol# / G#	5	830,61
La / A	5	880,00
La# / A#	5	932,33
Si / B	5	987,77
Do / C	6	1046,50
Do# / C#	6	1108,73
Re / D	6	1174,66
Re# / D#	6	1244,51
Mi / E	6	1318,51
Fa / F	6	1396,91
Fa# / F#	6	1479,98
Sol / G	6	1567,98
Sol# / G#	6	1661,22
La / A	6	1760,00
La# / A#	6	1864,66
Si / B	6	1975,53
Do / C	7	2093,01
Do# / C#	7	2217,46
Re / D	7	2349,32
Re# / D#	7	2489,02

Mi / E	7	2637,02
Fa / F	7	2793,83
Fa# / F#	7	2959,96
Sol / G	7	3135,96
Sol# / G#	7	3322,44
La / A	7	3520,00
La# / A#	7	3729,31
Si / B	7	3951,07
Do / C	8	4186,01
Do# / C#	8	4434,92
Re / D	8	4698,64
Re# / D#	8	4978,03
Mi / E	8	5274,04
Fa / F	8	5587,65
Fa# / F#	8	5919,91
Sol / G	8	6271,93
Sol# / G#	8	6644,88
La / A	8	7040,00
La# / A#	8	7458,62
Si / B	8	7902,13
Do / C	9	8372,02
Do# / C#	9	8869,84
Re / D	9	9397,27
Re# / D#	9	9956,06
Mi / E	9	10548,08
Fa / F	9	11175,30
Fa# / F#	9	11839,82
Sol / G	9	12543,86
Sol# / G#	9	13289,75
La / A	9	14080,00
La# / A#	9	14917,24
Si / B	9	15804,26