PRÁCTICAS DE ARDUINO

Práctica nº 10: Música con Arduino

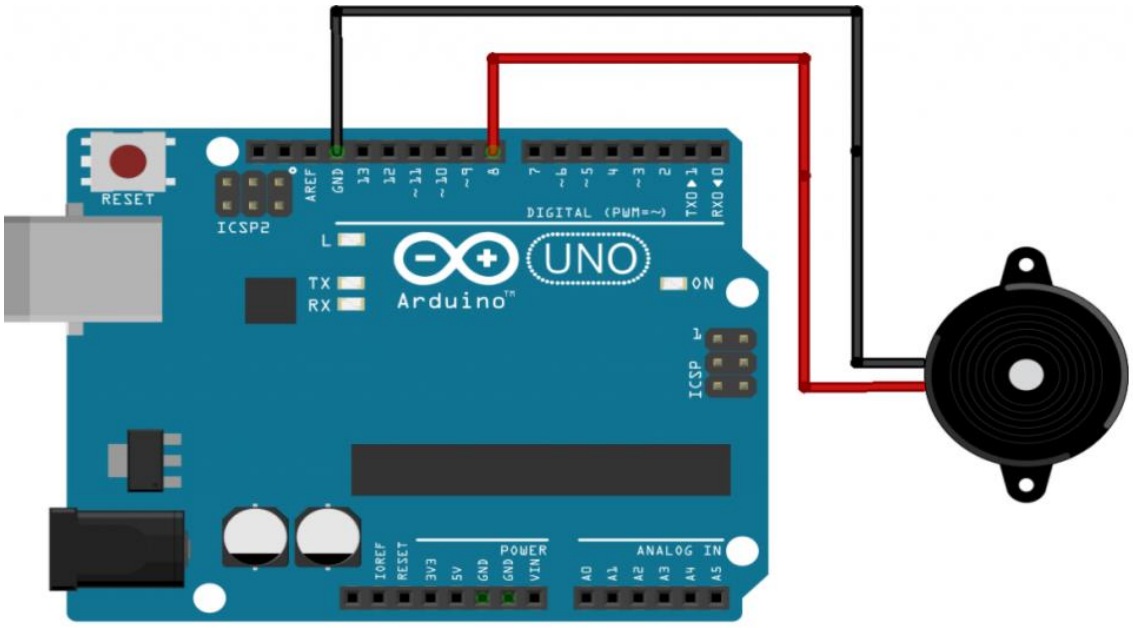
Esta práctica consiste en programar una melodía.

MATERIALES

- Arduino UNO.
- Cable USB tipo A-B.
- 1 Buzzer
- Cables de conexión.

MONTAJE

Para hacerlo sonar utilizaremos los pines 8 y GND.



PROGRAMACIÓN

Utilizaremos las siguientes instrucciones:

TONE

tone(pin, frecuencia);

tone(pin, frecuencia, tiempo);

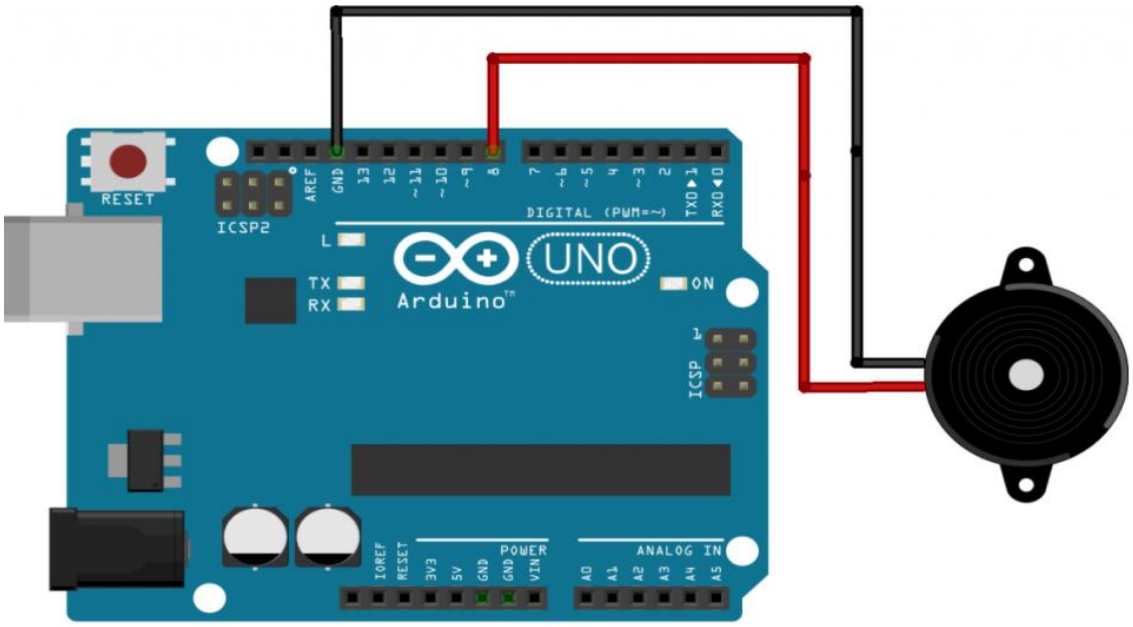
Si no se especifica duración suena indefinidamente hasta que se manda otro sonido o bien se manda la orden de silenciar noTone(pin). El sonido se ejecuta en background, eso quiere decir que el programa manda la orden de que suene y sigue ejecutando las órdenes siguientes. Si pones varios sonidos seguidos, se solaparán y no sonarán bien. Hay que incluir un **delay** de la misma duración que la nota para que la siguiente suene en su lugar correcto.

NOTA

nota(frecuencia, duración);

Estas son las frecuencias de las notas del piano (en Hz).

Octavas	Cero	1	2	3	4	5	6	7	8
DO	33	65	131	262	523	1047	2093	4186	
DO#	35	69	139	277	554	1109	2217		
RE	37	73	147	294	587	1175	2349		
RE#	39	78	156	311	622	1245	2489		
MI	41	82	165	330	659	1319	2637		
FA	44	87	175	349	698	1397	2794		
FA#	46	92	185	370	740	1480	2960		
SOL	49	98	196	392	784	1568	3136		
SOL#	52	104	208	415	831	1661	3322		
LA	28	55	110	220	440	880	1760	3520	
LA#	29	58	117	233	466	932	1865	3729	
SI	31	62	123	247	494	988	1976	3951	



EJEMPLO DE CÓDIGO

Una vez realizado el montaje para comprobar su correcto funcionamiento, cargamos en nuestro IDE Arduino el código fuente procedente de **EJEMPLOS<DIGITAL<TONEMELODY** que podemos ver a continuación:

```
/*
  Melody

  http://www.arduino.cc/en/Tutorial/Tone
*/

#include "pitches.h"

// Cargamos el fichero pitches.h, que es un librería (programa) de
Arduino que incluye todas las posibles notas que se podrán reproducir
en nuestro programa (NOTE_B0, NOTE_C1…)

int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};

int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};
// En el array noteDurations[ ] vamos a poder decidir la duración de
cada una de estas notas, donde un 4 será un cuarto de tono, un 8 un
octavo de tono, y así sucesivamente.
```

```
void setup() {
  // Iteración  sobre las notas de la melodía:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // Para calcular la duración de la nota, dividimos un segundo
dividido por el tipo de nota.
    //Por ejemplo un cuarto de nota = 1000 / 4, un octavo de nota =
1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(8, melody[thisNote], noteDuration);

    // Para distinguir las notas, se establece un tiempo mínimo entre
ellas.
    // La duración de la nota + 30% parece funcionar bien:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // Deja de tocar el tono:
    noTone(8);
  }
}

void loop() {
  // Ya no hace falta repetir la melodía.
}
```

En primer lugar nos encontramos la inclusión de la librería **pitches.h**, ya contenida en Arduino, y que comprende todas las posibles variantes de notas. En esta librería lo que se realiza es una asignación de una nota concreta a una freciuencia determinada, el **rango de frecuencias va de 31 Hz a 65535 Hz.** Estas notas (NOTE _C4, NOTE_G3…) son las que se citan en el array de tipo entero **melody[ ].** Ambas instrucciones serán las que nos permitirán reproducir los distintos sonidos.
El contenido de la librería **pitches.h** sería el siguiente:

```
/*************************************************
  Public Constants
 *************************************************/

 #define NOTE_B0  31
 #define NOTE_C1  33
 #define NOTE_CS1 35
 #define NOTE_D1  37
 #define NOTE_DS1 39
 #define NOTE_E1  41
 #define NOTE_F1  44
 #define NOTE_FS1 46
 #define NOTE_G1  49
 #define NOTE_GS1 52
 #define NOTE_A1  55
 #define NOTE_AS1 58
 #define NOTE_B1  62
 #define NOTE_C2  65
 #define NOTE_CS2 69
 #define NOTE_D2  73
 #define NOTE_DS2 78
 #define NOTE_E2  82
```

```
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
```

```
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

Cabe destacar las siguientes aspectos del código anterior:

- La estructura de control **for ,** está programado para repetirse 8 veces, de 0 a 7, tantas como notas tenemos en nuestra melodía, incluyendo el silencio musical o pausa.
- La función ***tone()*** tiene tres argumentos:
    - La salida digital donde hemos conectado uno de los terminales del buzzer, en este caso el **pin** digital nº 8.
    - La **frecuencia o nota** correspondiente según el paso del bucle **for**. ***Los rangos de la función tone son de 31 Hz a 65535 Hz.***
    - La **duración** de la nota que la hemos calculado en la instrucción anterior.
- Hacemos una pausa o **delay** tras reproducir la nota, será la pausa entre notas, calculada en la instrucción anterior.
- Para terminar detenemos la reproducción de notas con la función ***noTone()*** y el pin digital de nuestro buzzer.

El siguiente, también muy básico, emplea un array con frecuencias que recorremos secuencialmente para realizar un barrido que aproxima las distintas notas musicales.

```
const int pinBuzzer = 8;

const int tonos[] = {261, 277, 294, 311, 330, 349, 370, 392, 415, 440,
466, 494};
const int countTonos = 12;

void setup()
{
}

void loop()
{
 for (int iTono = 0; iTono < countTonos; iTono++)
 {
 tone(pinBuzzer, tonos[iTono]);
 delay(500);
 }
 noTone(pinBuzzer);
 delay (5000);
}
```

Para no almacenar todas las frecuencias y tener que escribir la función tone en nuestro código cada vez que queramos una nueva nota, utilizaremos el bucle **for**. Almacenaremos sólo el valor de la frecuencia inicial, y las sucesivas notas tendrán la frecuencia de la anterior multiplicada por 1,059. De este modo escribiremos una sola

vez la función para hacerlo sonar y un bucle for será el encargado de ir incrementando el valor de la frecuencia.

```
int pinaltavoz = 8;
int frecuencia=220;    // frecuencia correspondiente a la nota La
int contador;          // variable para el contador
float m=1.059;         // constante para multiplicar frecuencias

void setup()
{
}

void loop()
{
    for(contador=0,frecuencia=220;contador<12;contador++)
    {
        frecuencia=frecuencia*m;     // actualiza la frecuencia
        tone(pinaltavoz,frecuencia); // emite el tono
        delay(1500);                 // lo mantiene 1.5 segundos
        noTone(pinaltavoz);          // para el tono
        delay(500);                  // espera medio segundo
    }
}
```

Ahora pasamos a reproducir música, pero de la que recuerda a los videojuegos con **música de 8 bits.**

Para conseguirlo utilizaremos una función auxiliar que llamaremos **nota** con la siguiente estructura:

Nota (frecuencia,duración)

Esta instrucción nos ahorrará escribir decenas de veces la **función delay** para indicar el tiempo que debe durar la nota. Cada vez que llamemos a esa función se ejecutará esta parte del código.

Equivalencia entre notas musicales y frecuencias:

OCTAVA -2		
NOTA	Nº	HZ
DO	0	8.18
DO#	1	8.66
RE	2	9.18
RE#	3	9.72
MI	4	10.30
FA	5	10.91
FA#	6	11.56
SOL	7	12.25
SOL#	8	12.98
LA	9	13.75
LA#	10	14.57
SI	11	15.43

OCTAVA -1		
NOTA	Nº	HZ
DO	12	16.35
DO#	13	17.32
RE	14	18.35
RE#	15	19.45
MI	16	20.60
FA	17	21.83
FA#	18	23.12
SOL	19	24.50
SOL#	20	25.96
LA	21	27.50
LA#	22	29.14
SI	23	30.87

OCTAVA 0		
NOTA	Nº	HZ
DO	24	32.70
DO#	25	34.65
RE	26	36.71
RE#	27	38.89
MI	28	41.20
FA	29	43.65
FA#	30	46.25
SOL	31	49.00
SOL#	32	51.91
LA	33	55.00
LA#	34	58.27
SI	35	61.74

OCTAVA 1		
NOTA	Nº	HZ
DO	36	65.41
DO#	37	69.30
RE	38	73.42
RE#	39	77.78
MI	40	82.41
FA	41	87.31
FA#	42	92.50
SOL	43	98.00
SOL#	44	103.83
LA	45	110.00
LA#	46	116.54
SI	47	123.47

OCTAVA 2		
NOTA	Nº	HZ
DO	48	130.81
DO#	49	138.59
RE	50	146.83
RE#	51	155.56
MI	52	164.81
FA	53	174.61
FA#	54	185.00
SOL	55	196.00
SOL#	56	207.65
LA	57	220.00
LA#	58	233.08
SI	59	246.94

OCTAVA 3		
NOTA	Nº	HZ
DO	60	261.63
DO#	61	277.18
RE	62	293.66
RE#	63	311.13
MI	64	329.63
FA	65	349.23
FA#	66	369.99
SOL	67	392.00
SOL#	68	415.30
LA	69	440.00
LA#	70	466.16
SI	71	493.88

Aquí tenéis varios fragmentos de canciones para practicar:

Melodía Star WARS (Tema principal)

```
int spk=8;                                          // altavoz a GND
y pin 8
int c[5]={131,262,523,1046,2093};       // frecuencias 4 octavas de Do
int cs[5]={139,277,554,1108,2217};      // Do#
int d[5]={147,294,587,1175,2349};       // Re
int ds[5]={156,311,622,1244,2489};    // Re#
int e[5]={165,330,659,1319,2637};       // Mi
int f[5]={175,349,698,1397,2794};       // Fa
int fs[5]={185,370,740,1480,2960};     // Fa#
int g[5]={196,392,784,1568,3136};     // Sol
int gs[5]={208,415,831,1661,3322};   // Sol#
int a[5]={220,440,880,1760,3520};       // La
int as[5]={233,466,932,1866,3729};    // La#
int b[5]={247,494,988,1976,3951};       // Si

void nota(int a, int b);            // declaración de la función
auxiliar. Recibe dos números enteros

void setup()
{
nota(d[1],150);noTone(spk);delay(50);
nota(d[1],150);noTone(spk);delay(50);
nota(d[1],150);noTone(spk);delay(50);
nota(g[1],900);noTone(spk);delay(150);
nota(d[2],900);noTone(spk);delay(50);
nota(c[2],150);noTone(spk);delay(50);
nota(b[1],150);noTone(spk);delay(50);
nota(a[1],150);noTone(spk);delay(50);
nota(g[2],900);noTone(spk);delay(150);
```

```
nota(d[2],900);noTone(spk);delay(100);
nota(c[2],150);noTone(spk);delay(50);
nota(b[1],150);noTone(spk);delay(50);
nota(a[1],150);noTone(spk);delay(50);
nota(g[2],900);noTone(spk);delay(150);
nota(d[2],900);noTone(spk);delay(100);
nota(c[2],150);noTone(spk);delay(50);
nota(b[1],150);noTone(spk);delay(50);
nota(c[2],150);noTone(spk);delay(50);
nota(a[1],1200);noTone(spk);delay(2000);

  }

void nota(int frec, int t)
{
    tone(spk,frec);      // suena la nota frec recibida
    delay(t);               // para después de un tiempo t
}

void loop()
{
}
```

Melodía Star WARS (Marcha del imperio)

```
int spk=8;                                          // altavoz a GND
y pin 8
int c[5]={131,262,523,1046,2093};       // frecuencias 4 octavas de Do
int cs[5]={139,277,554,1108,2217};      // Do#
int d[5]={147,294,587,1175,2349};       // Re
int ds[5]={156,311,622,1244,2489};    // Re#
int e[5]={165,330,659,1319,2637};       // Mi
int f[5]={175,349,698,1397,2794};       // Fa
int fs[5]={185,370,740,1480,2960};     // Fa#
int g[5]={196,392,784,1568,3136};     // Sol
int gs[5]={208,415,831,1661,3322};   // Sol#
int a[5]={220,440,880,1760,3520};       // La
int as[5]={233,466,932,1866,3729};    // La#
int b[5]={247,494,988,1976,3951};       // Si

void nota(int a, int b);            // declaración de la función
auxiliar. Recibe dos números enteros

void setup()
{
nota(g[2],500);noTone(spk);delay(100);
nota(g[2],500);noTone(spk);delay(100);
nota(g[2],500);noTone(spk);delay(100);
nota(ds[2],500);noTone(spk);delay(1);
nota(as[2],125);noTone(spk);delay(25);
nota(g[2],500);noTone(spk);delay(100);
nota(ds[2],500);noTone(spk);delay(1);
nota(as[2],125);noTone(spk);delay(25);
nota(g[2],500);
noTone(spk);delay(2000);

  }

void nota(int frec, int t)
```

```
{
    tone(spk,frec);      // suena la nota frec recibida
    delay(t);                // para después de un tiempo t
}

void loop()
{
}
```

Mario Bross

```
int portSpeak(9);  //porta ligada no speaker

//melodia MARIO BROSS
int melodia[] =
{660,660,660,510,660,770,380,510,380,320,440,480,450,430,380,660,760,8
60,700,760,660,520,580,480,510,380,320,440,480,450,430,380,660,760,860
,700,760,660,520,580,480,500,760,720,680,620,650,380,430,500,430,500,5
70,500,760,720,680,620,650,1020,1020,1020,380,500,760,720,680,620,650,
380,430,500,430,500,570,585,550,500,380,500,500,500,500,760,720,680,62
0,650,380,430,500,430,500,570,500,760,720,680,620,650,1020,1020,1020,3
80,500,760,720,680,620,650,380,430,500,430,500,570,585,550,500,380,500
,500,500,500,500,500,500,580,660,500,430,380,500,500,500,500,580,660,8
70,760,500,500,500,500,580,660,500,430,380,660,660,660,510,660,770,380
};

//duración de cada nota
int duracioncadanota[] =
{100,100,100,100,100,100,100,100,100,100,100,80,100,100,100,80,50,100,
80,50,80,80,80,80,100,100,100,100,80,100,100,100,80,50,100,80,50,80,80
,80,80,100,100,100,100,150,150,100,100,100,100,100,100,100,100,100,100
,150,200,80,80,80,100,100,100,100,100,150,150,100,100,100,100,100,100,
100,100,100,100,100,100,100,100,100,100,100,150,150,100,100,100,100,10
0,100,100,100,100,100,150,200,80,80,80,100,100,100,100,100,150,150,100
,100,100,100,100,100,100,100,100,100,100,100,100,60,80,60,80,80,80,80,
80,80,60,80,60,80,80,80,80,80,60,80,60,80,80,80,80,80,80,100,100,100,1
00,100,100,100};

void setup() {
  //for para tocar las 156 notas comenzando en el 0 hasta 156 ++
incrementadolo
        for (int nota = 0; nota < 156; nota++) {

              int duracionnota = duracioncadanota[nota];
              tone(portSpeak, melodia[nota],duracionnota);
  //pausa después de las notas
              int pausadespuesnotas[]
={150,300,300,100,300,550,575,450,400,500,300,330,150,300,200,200,150,
300,150,350,300,150,150,500,450,400,500,300,330,150,300,200,200,150,30
0,150,350,300,150,150,500,300,100,150,150,300,300,150,150,300,150,100,
220,300,100,150,150,300,300,300,150,300,300,300,100,150,150,300,300,15
0,150,300,150,100,420,450,420,360,300,300,150,300,300,100,150,150,300,
300,150,150,300,150,100,220,300,100,150,150,300,300,300,150,300,300,30
0,100,150,150,300,300,150,150,300,150,100,420,450,420,360,300,300,150,
300,150,300,350,150,350,150,300,150,600,150,300,350,150,150,550,325,60
0,150,300,350,150,350,150,300,150,600,150,300,300,100,300,550,575};
              delay(pausadespuesnotas[nota]);}

              noTone(portSpeak);
}
```

```
void loop() {

}
```

```
/* Programa que reproduce la melodia de "Piratas del Caribe */

void setup(){
 pinMode (9, OUTPUT); //pin configurado como salida
}
// funcion = tone(pin, frequency, duration)

void loop() {
  tone(9,293.66,200);
  delay(200);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,200);
  delay(200);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,200);
  delay(200);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,200);
  delay(200);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,200);
  delay(200);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,200);
  delay(200);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,100);
  delay(100);
  tone(9,293.66,200);
  delay(200);
  tone(9,293.66,100);
  delay(100);
```

```
tone(9,293.66,200);
delay(200);
tone(9,293.66,100);
delay(100);
tone(9,293.66,200);
delay(200);
tone(9,293.66,100);
delay(100);
tone(9,293.66,100);
delay(100);
tone(9,440,100);
delay(100);
tone(9,523.25,100);
delay(100);
tone(9,587.33,100);
delay(200);
tone(9,587.33,100);
delay(200);
tone(9,587.33,100);
delay(100);
tone(9,659.25,100);
delay(100);
tone(9,698.45,100);
delay(200);
tone(9,698.45,100);
delay(200);
tone(9,698.45,100);
delay(100);
tone(9,783.99,100);
delay(100);
tone(9,659.25,100);
delay(200);
tone(9,659.25,100);
delay(200);
tone(9,587.33,100);
delay(100);
tone(9,523.25,100);
delay(100);
tone(9,523.25,100);
delay(100);
tone(9,587.33,100);
delay(300);
tone(9,440,100);
delay(100);
tone(9,523.25,100);
delay(100);
tone(9,587.33,100);
delay(200);
tone(9,587.33,100);
delay(200);
tone(9,587.33,100);
delay(100);
tone(9,659.25,100);
```

```
  delay(100);
  tone(9,698.45,100);
  delay(200);
  tone(9,698.45,100);
  delay(200);
  tone(9,698.45,100);
  delay(100);
  tone(9,783.99,100);
  delay(100);
  tone(9,659.25,100);
  delay(200);
  tone(9,659.25,100);
  delay(200);
  tone(9,587.33,100);
  delay(100);
  tone(9,523.25,100);
  delay(100);
  tone(9,587.33,100);
  delay(400);
  tone(9,440,100);
  delay(100);
  tone(9,523.25,100);
  delay(100);
  tone(9,587.33,100);
  delay(200);
  tone(9,587.33,100);
  delay(200);
  tone(9,587.33,100);
  delay(100);
  tone(9,698.45,100);
  delay(100);
  tone(9,783.99,100);
  delay(200);
  tone(9,783.99,100);
  delay(200);
  tone(9,783.99,100);
  delay(100);
  tone(9,880,100);
  delay(100);
  tone(9,932.33,100);
  delay(200);
  tone(9,932.33,100);
  delay(200);
  tone(9,880,100);
  delay(100);
  tone(9,783.99,100);
  delay(100);
  tone(9,880,100);
  delay(100);
  tone(9,587.33,100);
  delay(300);
  tone(9,587.33,100);
  delay(100);
```

```
tone(9,659.25,100);
delay(100);
tone(9,698.45,100);
delay(200);
tone(9,698.45,100);
delay(200);
tone(9,783.99,100);
delay(200);
tone(9,880,100);
delay(100);
tone(9,587.33,100);
delay(300);
tone(9,587.33,100);
delay(100);
tone(9,698.45,100);
delay(100);
tone(9,659.25,100);
delay(200);
tone(9,659.25,100);
delay(200);
tone(9,698.45,100);
delay(100);
tone(9,587.33,100);
delay(100);
tone(9,659.25,100);
delay(400);
tone(9,880,100);
delay(100);
tone(9,1046.50,100);
delay(100);
tone(9,1174.66,100);
delay(200);
tone(9,1174.66,100);
delay(200);
tone(9,1174.66,100);
delay(100);
tone(9,1318.51,100);
delay(100);
tone(9,1396.91,100);
delay(200);
tone(9,1396.91,100);
delay(200);
tone(9,1396.91,100);
delay(100);
tone(9,1567.98,100);
delay(100);
tone(9,1318.51,100);
delay(200);
tone(9,1318.51,100);
delay(200);
tone(9,1174.66,100);
delay(100);
tone(9,1046.50,100);
```

```
  delay(100);
  tone(9,1046.50,100);
  delay(100);
  tone(9,1174.66,100);
  delay(300);
  tone(9,880,100);
  delay(100);
  tone(9,1046.50,100);
  delay(100);
  tone(9,1174.66,100);
  delay(200);
  tone(9,1174.66,100);
  delay(200);
  tone(9,1174.66,100);
  delay(100);
  tone(9,1318.51,100);
  delay(100);
  tone(9,1396.91,100);
  delay(200);
  tone(9,1396.91,100);
  delay(200);
  tone(9,1396.91,100);
  delay(100);
  tone(9,1567.98,100);
  delay(100);
  tone(9,1318.51,100);
  delay(200);
  tone(9,1318.51,100);
  delay(200);
  tone(9,1174.66,100);
  delay(100);
  tone(9,1046.50,100);
  delay(100);
  tone(9,1174.66,100);
  delay(400);
  tone(9,880,100);
  delay(100);
  tone(9,1046.50,100);
  delay(100);
  tone(9,1174.66,100);
  delay(200);
  tone(9,1174.66,100);
  delay(200);
  tone(9,1174.66,100);
  delay(100);
  tone(9,1396.91,100);
  delay(100);
  tone(9,1567.98,100);
  delay(200);
  tone(9,1567.98,100);
  delay(200);
  tone(9,1567.98,100);
  delay(100);
```

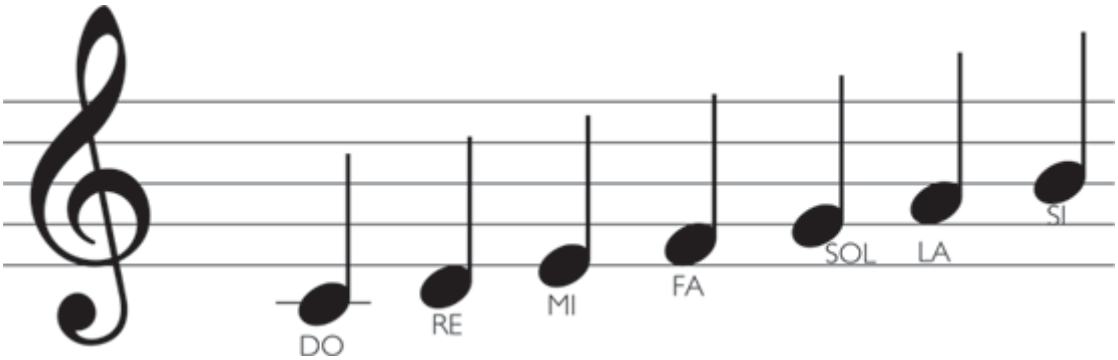
```
  tone(9,1760,100);
  delay(100);
  tone(9,1864.66,100);
  delay(200);
  tone(9,1864.66,100);
  delay(200);
  tone(9,1760,100);
  delay(100);
  tone(9,1567.98,100);
  delay(100);
  tone(9,1760,100);
  delay(100);
  tone(9,1174.66,100);
  delay(300);
  tone(9,1174.66,100);
  delay(100);
  tone(9,1318.51,100);
  delay(100);
  tone(9,1396.91,100);
  delay(200);
  tone(9,1396.91,100);
  delay(200);
  tone(9,1567.98,100);
  delay(200);
  tone(9,1760,100);
  delay(100);
  tone(9,1174.66,100);
  delay(300);
  tone(9,1174.66,100);
  delay(100);
  tone(9,1396.91,100);
  delay(100);
  tone(9,1318.51,100);
  delay(200);
  tone(9,1318.51,100);
  delay(200);
  tone(9,1174.66,100);
  delay(100);
  tone(9,1108.73,100);
  delay(100);
  tone(9,1174.66,100);
  delay(200);
  tone(9,1174.66,100);
  delay(200);
  tone(9,1318.51,100);
  delay(200);
  tone(9,1396.91,100);
  delay(200);
  tone(9,1396.91,100);
  delay(100);
  tone(9,1396.91,100);
  delay(100);
  tone(9,1567.98,100);
```

```
  delay(200);
  tone(9,1760,300);
  delay(400);
  tone(9,1396.91,100);
  delay(100);
  tone(9,1174.66,100);
  delay(100);
  tone(9,880,300);
  delay(600);
  tone(9,1864.66,300);
  delay(400);
  tone(9,1396.91,100);
  delay(100);
  tone(9,1174.66,100);
  delay(100);
  tone(9,932.33,300);
  delay(600);
  tone(9,587.33,100);
  delay(100);
  tone(9,440,100);
  delay(200);
  tone(9,587.33,100);
  delay(300);
  tone(9,554.36,100);
  delay(400);
  tone(9,1567.98,100);
  delay(100);
  tone(9,1567.98,100);
  delay(100);


  }
```

Ejercicios propuestos

1. Comprueba que al cargar el siguiente programa “ESCALA MUSICAL DE 7 NOTAS”, el zumbador reproduce la escala musical.




```
#define BUZZER 8
int notes[] = { 524, 588, 660, 699, 785, 881, 989 };
void setup()
```

```
{
pinMode(BUZZER, OUTPUT);
}
void loop()
{
for (int i = 0; i < 7; i++)
{
tone(BUZZER, notes[i], 1000);
delay(1000);
}
delay(1000);
}
```

2. Escribe una melodía y haz un programa que vaya haciendo sonar frecuencias

3. Elige una canción que te guste, busca sus notas musicales, realiza las equivalencia de esas notas a frecuencias y haz un programa que vaya haciendo sonar frecuencias en Arduino.