M.A.D.E.

Intermediate E-textiles Guides

for Music | Art | Design | Experiences

Making Music

Please cite this work as Fields, D. A., Amely, J., Jayathirtha, G., Lindberg, L., Lui, D. & Kafai, Y. B. (2019). *M.A.D.E. Intermediate E-Textiles Guide: Making Music.* Available online at http://exploringcs.org/e-textiles/modules.

Emma Eccles Jones College of Education and Human Services Utah State University

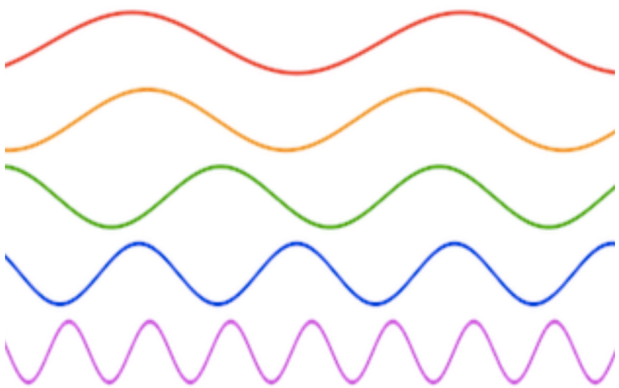
Graduate School of Education Penn GSE

ECS Exploring Computer Science

What you need to know first:

This module assumes that users have basic experience with making lighting patterns in Arduino (i.e., with digitalWrite() and delay() commands). For more introductory material, see the Exploring Computer Science e-textiles curriculum unit, Buechley & Lui Sew Electric, or other introductory Arduino guides. Other ECS E-Textiles modules as well as supporting code samples that may be referenced in this text can be found through http://exploringcs.org/e-textiles/modules.

Table of Contents

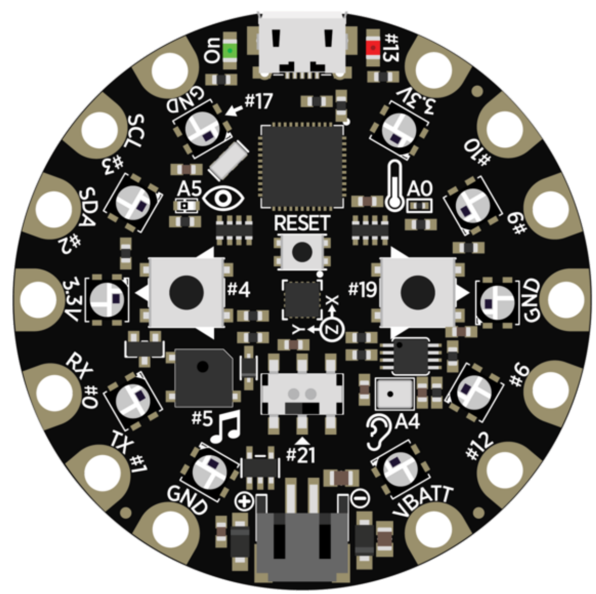


1. Introduction to Music & Sound

Sound makes up music, but what is sound? Nothing but waves! We humans can hear these waves when they are within 20 to 20,000 **hertz** (Hz), which is how the **frequency** of the waves is measured.

In the picture on the right, the lowest frequency (with the longest waves) is shown in red. It has a low number of hertz and a low pitch to our ear. The purple frequency has a high number of hertz (shorter waves), and has a high pitch to our ears!

> **Longer the wave, lower the hertz = Lower Pitch**
> **Shorter the wave, higher the hertz = Higher Pitch**
> **Hertz = numerical value for frequency of waves (how many waves in a period of time)**



So then if sound can be described in waves (measured in hertz), what is music? If you break it down to the very basics of what *makes* music, you get meaningfully arranged **notes**. A note holds 2 pieces of important information: the **pitch** (measured in hertz), and the **duration** (how long it sounds).

> **Note = Pitch + Duration**

1.1 Programming a Note

To code sounds on the Circuit Playground, we can use the speaker on pin #5. You can find it by the music note next to the black box on the bottom left quadrant. (On other microcontrollers you may need to attach a speaker to a pin, but the Circuit Playground already has one attached.)

If we want the Circuit Playground to play notes, we have to give it the two pieces of information needed to create one—the pitch and the duration (aka **pitch = tone** and **duration = delay**). Look at the Example | Play a Note code below. There are two new functions for us to use, `tone()` and `notone()`. This is how we will program a single note to play and then stop playing.

EXAMPLE | Play a Note

```
int speaker = 5;

void setup(){

}

void loop(){
  tone (speaker, 440);
  delay(150);
  notone(speaker);
  delay(1000);
}
```

1.2 CHALLENGE | Jingle

Now it is your turn! Start a New sketch in Arduino, and save it as music_aNote. Use Save As… between challenges to save multiple sketches at each point along your learning path.

- ❏ **EASIEST** Type in the code above and make sure it works to play a single note.
- ❏ **EASY** Change your code to make your tone higher. Do you remember the name for this measurement?
- ❏ **EASY** Add another note into your code and make it lower than your first note. (*HINT:* you need all 4 lines of code to make another note.)
- ❏ **MEDIUM** Play with the rhythm of your two notes. The rhythm is made by how long you play a note and how long the pause between notes is. For both of those we are using `delay()`.
- ❏ **MEDIUM** Add more notes in different tones, some high, some low, and some the same. Add no more than 10 notes and at least 2 more, and keep them between 300 Hz & 1000 Hz. Be sure to vary the length (duration) of each note!
- ❏ **HARD** Arrange your composition in a way that is pleasing to the ear.
- ❏ **HARDEST** See if you can match the first set of notes in a common tune, such as "Twinkle Twinkle Little Star."

2. Breaking Down an Array

Writing notes out one-by-one like this is exceedingly tedious and prone to error. A much better way is to use two arrays and a `for()` loop to pair them together to play music. If you are not familiar with `for()` loops, take a look at Fading LEDs, Section 4 to get caught up.

You already know that a **variable** can store one value. An **array** can hold multiple values, so we could have an array that holds all the values for the notes in a song. For instance, we could write out 5 notes in a song, as in the array "notes" below,

Variable = One Value	Array = Multiple Values
`int  note1 = 300;`	`int Notes[] = {300, 320, 200, 150, 0};`

There are a couple of differences to notice here: The square brackets `[]` that are part of the name; the curly braces `{}` to hold your values; the commas `,` separating those values; and the actual values **`300, 320, 200, 150, 0`**.

`[]` = This lets the computer know that we are going to make an array.

`{}` = These hold all of the values that you put into an array. We say values because you could also do whole words, but we are only using numbers for now.

`,` = *Comma-separated values (CSV)* is a commonly-used computing term. By putting the comma in between each value, the computer knows where that value ends. CSV's can also be used in spreadsheets and lots of other programming languages!

2.1 TASK | Understanding Array Addresses

`Notes[0], Notes[1], Notes[2], Notes[3], Notes[4]`: You have 5 values in your **`Notes[]`** array: **`300, 320, 200, 150, 0`**. To reference the first spot, called an ***address*** like the address of your home, we use **`Notes[0];`**. This returns the first value on our list, **`300`**. *Zero Indexing* is what you call it when you start a place count with zero, and it is standard in many programming languages. What value do you get for **`Notes[3];`**? Remember to start counting the value addresses with zero!

Answer = 150

2.2 CHALLENGE: Identify the Address of Different Notes & Durations!

```
int MyPitches[] = {440, 880, 660, 881, 658, 523};
int MyDurations[] = {150, 137, 138, 139, 1100, 550};
```

EXAMPLE

```
MyPitches[0] = 440

MyDurations[4] = 1100
```

You have two arrays: **`MyPitches[]`** and **`MyDurations[]`**. Find the values *or* the addresses in the Arrays. Fill in the correct numbers!

Values	Addresses
MyPitches[0] = ___440___	MyDurations[__] = 1100
MyDurations[0] = _________	MyPitches[__] = 523
MyDurations[3] = _________	MyDurations[__] = 137
MyPitches[3] = _________	MyPitches[__] = 660
MyDurations[5] = _________	MyDurations[__] = 139
MyPitches[5] = _________	MyPitches[__] = 440

Answers: Values = 440, 150, 139, 881, 550, 523 | Addresses: 4, 5, 1, 2, 3, 0

2.3 Writing a Musical Array

We could use the same 4 lines of code to play a note as [before](), by directly substituting in **`tone(speaker, MyPitches[0]);`** for the line **`tone(speaker, 440);`** and **`delay(MyDurations[0]);`** for **`delay(150);`**. But then why go through all of the hassle of setting up an array?

Instead, let's take our two arrays, **`MyPitches[]`** and **`MyDurations[]`**, and use a **`for()`** loop to pair the note values together to create and play notes. Find a guide for **`for()`** loops at Fading LEDs, Section 4. Take a look at the code below to see how to do that.

EXAMPLE | Music Array

```
int speaker = 5;
int MyPitches[] = {440, 880, 660, 881, 658, 523};
int MyDurations[] = {150, 137, 138, 139, 1100, 550};

void setup(){

}

void loop(){
  for (int i = 0; i < 6; i = i++){
    tone (speaker, MyPitches[i]);
    delay(MyDurations[i]);
    notone(speaker);
    delay(10);
  }
}
```

The first time through the `for()` loop, what note (pitch and duration) will be played?
Notice that we set `i` to `0`. Keep in mind that the address in the array starts with `0`.
Answer = The note played will have a pitch of 440 and a duration of 150.

Can you figure out what the 5th time through the `for()` loop plays?
Why is our condition `i` ***LESS THAN*** `6` if we need to play 6 notes?
Answer = The fifth time through plays 658/1100, and the sixth time through plays 523/550, respectively. The address locations start at 0, so we count, 0 – 1 – 2 – 3 – 4 – 5, which equals 6 numbers, but since we start on 0, is less than 6.

2.4 CHALLENGE | Mystery Melodies

We have 5 array sets that each play a Mystery Melody if they are coded into a `for()` loop correctly. See how many Melodies you can decipher!
*HINT: You can edit the code above (Example | Music Array) and add the values of the pitches and durations to the arrays. Be sure to adjust your `for()` loop so that it plays the right number of notes for each song!

- ❏ Melody 1
  - Melody 1 pitches = 196, 196, 196, 262, 392, 349, 330, 294, 523, 392, 349, 330, 294, 523, 392, 349, 330, 349, 294
  - Melody 1 durations = 184, 184, 184, 1100, 1100, 184, 184, 184, 1100, 550, 184, 184, 184, 1100, 550, 184, 184, 184, 1100
- ❏ Melody 2
  - Melody 2 pitches = 494, 659, 784, 740, 659, 988, 880, 740, 659, 784, 740, 587, 698, 494, 494
  - Melody 2 durations = 550, 825, 275, 550, 1100, 550, 1650, 1650, 825, 275, 550, 1100, 550, 1650, 1100
- ❏ Melody 3
  - Melody 3 pitches = 294, 294, 294, 330, 370, 370, 330, 370, 392, 440, 587, 587, 587, 440, 440, 440, 370, 370, 370, 294, 294, 294, 440, 392, 370, 330, 294
  - Melody 3 durations = 825, 825, 550, 275, 825, 550, 275, 550, 275, 1650, 275, 275, 275, 275, 275, 275, 275, 275, 275, 275, 275, 275, 550, 275, 550, 275, 825

- Melody 4

  Melody 4 pitches = 1760, 494, 523, 0, 392, 392, 587, 523, 440, 440, 440, 494, 523, 1760, 494, 523, 0, 392, 659, 587, 523,
  587, 659, 659, 698, 659, 587, 523

  Melody 4 durations = 275, 275, 1375, 275, 275, 275, 1925, 550, 550, 550, 550, 275, 1375, 275, 275, 1375, 275, 275, 275,
  1925, 275, 275, 550, 550, 550, 275, 275, 2200

- Melody 5

  Melody 5 pitches = 196, 0, 196, 233, 262, 196, 0, 196, 175, 185, 196, 0, 196, 233, 262, 196, 0, 196, 0, 0, 932, 784, 587, 0,
  932, 784, 554, 0, 932, 784, 523, 0, 466, 523, 0, 0, 0, 0

  Melody 5 durations = 400, 200, 600, 400, 400, 400, 200, 600, 400, 400, 400, 200, 600, 400, 400, 400, 200, 600, 400, 500,
  200, 200, 1200, 500, 200, 200, 1200, 500, 200, 200, 1200, 500, 200, 200, 400, 400, 400, 500

Find the solutions on the last page, but no cheating—it takes the fun out of it!

3. Converting Your Own Sheet Music

Although it is fun to tinker around with composing our own music with the Circuit Playground, it is more interesting to play our favorite songs. This section will take you through several steps to convert a song from sheet music to Arduino. **Step one**: decipher your song's pitches (tones); **step two**: decipher your song's pitch durations; and **step three**: code the two arrays together(pitch+duration = note) to play your song.

Step zero is to find some simple sheet music. Use a search engine and type in **"[**song you want**] easy piano version"** then click on **Images**.

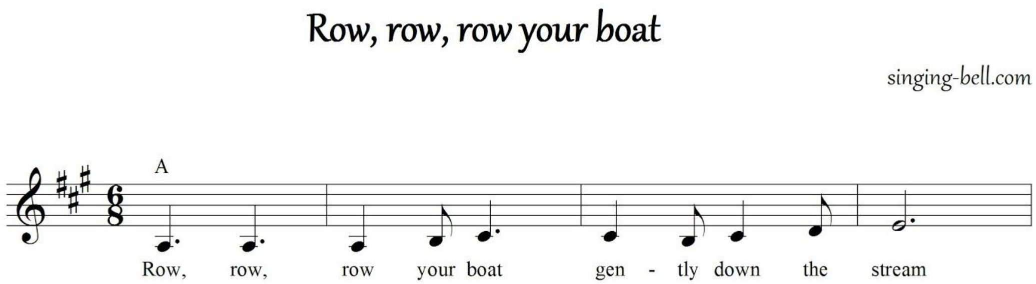
Tips:

- Think about the most recognizable part of your song. It might be at the beginning or it might be in another part of the song (like the chorus or something that repeats). You don't have to code the entire song to play your favorite part!
- The top line of the notes is usually the melody. Look at the top line of your favorite section to figure out which notes to translate. You can only play one note at a time on the small speakers we use with Arduino.

The code for your music will look similar to the code shown here. It may look a bit difficult right now, but we will work through each step. This code is shown again at the end of the module.

EXAMPLE | Converted Sheet Music Code (Row Your Boat)

Full “Row Your Boat” Starter Code (found at http://www.exploringcs.org/e-textiles/modules/supporting-code)



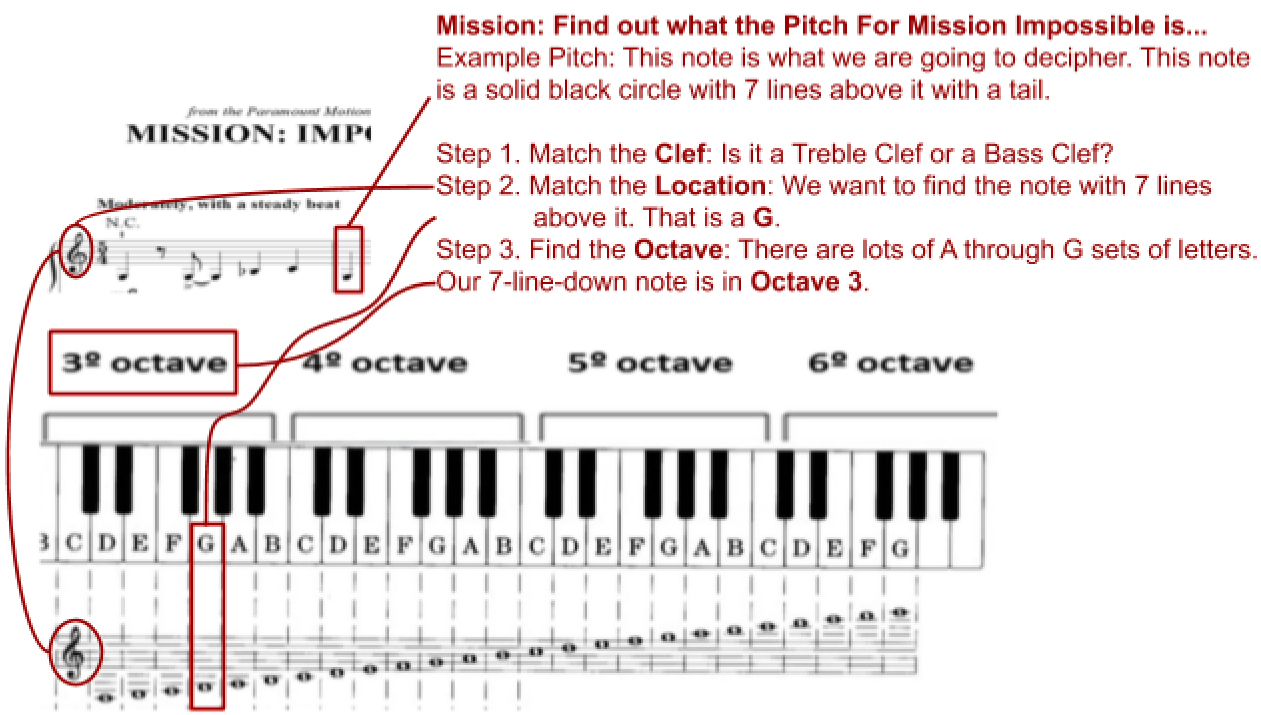

```
//Row Your Boat : This is an example of what your song will look like when done
int speaker = 5;
int pace = 2200;
//2200 is the length of a whole note. We can increase or decrease this to make the whole song slower or faster.
//Use float for the array, because float is for decimals instead of just whole numbers (integers).
//We are doing math right inside the array addresses and will get decimal point numbers back!
float MyDurations[] = {3.0/8.0, 3.0/8.0, 1.0/4.0, 1.0/8.0, 3.0/8.0, 1.0/4.0, 1.0/8.0, 1.0/4.0, 1.0/8.0,
3.0/4.0};
int MyPitches[] = {294, 294, 294, 330, 370, 370, 330, 370, 392, 440};

void setup(){
pinMode(speaker, OUTPUT);
}
void loop(){
   for (int i = 0; i < 10; i = i+1) { //Be sure to change the “i < 10” value to match your song
       tone (speaker, MyPitches[i]);
       delay(pace * MyDurations[i]);
       notone(speaker);
       delay(10);
   }
}
```

3.1 Decipher Note Pitch

With your sheet music in hand, you are ready to find the two pieces of information that make each note. Take a piece of scrap paper and make two columns on it labeled “Pitches” and “Durations”. We are going to go through the steps of how we got the “Row Your Boat” array of numbers for `MyPitches[]` and timings for `MyDurations[]`. We will start with the finding the pitch of each note. For right now, we are only concerned about the *location* of the note on the staff (the lines the notes are on).

Each pitch is a combination of location and octave. Each note is a combination of pitch and duration. One way you could think of it is **Note = (Location + Octave) + Duration**.



*Convert the pitch into notations. The example pitch will be converted into a notation as **G3**, where G is the pitch and 3 is the octave it belongs to.*

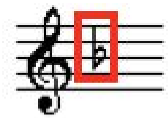
Special Musical Symbols:

- **Rest**: an interval of silence in a piece of music. The following picture shows the different rest symbols for different durations (rest longer/shorter).
    - **The pitch will be converted as the notation SILENCE (or tone = 0)**

- **Sharp (♯)**: higher in pitch by one semitone (half step)
    - When a pitch is higher by a semitone, for the notation you need to put an **S** between the pitch and the octave: **C4**→ **CS4**.
    - You might see sharp symbols at two different locations:
        1. Next to the clef: **make all of a particular pitch higher by a semitone**, regardless of its octave. In the example below, the squared space inside of the sharp symbol is located on the first line, which is the pitch F. Although it belongs to the 5th octave, you need to make all of the F pitches higher (e.g., FS1, FS2, FS3 …FS5) no matter if they belong to the 5th octave or not.

        2. Next to the music notes: **only raise the pitch of the notes in that measure** by one semitone. A measure is one of the spaces between the vertical lines. The example below shows the pitch **A4**. If the sharp symbol is right beside it, you need to raise only it to **AS4**.

- **Flat (♭)**: lower in pitch by one semitone
    - When a pitch is lower by a semitone, in the notation put an **F** between the pitch and the octave: **A4** → **AF4**.
    - Like the sharp symbols, you also might see flat symbols at two different locations:
        1. Next to the clef: **make all of a particular pitch lower by a semitone**, regardless of its octave. In the example below, the circle inside of the flat symbol is located on the third line, which is the pitch B. Although it belongs to the 4th octave, you need to make all the B pitches lower (BF1, BF2, BF3, BF4) no matter if they belong to the 4th octave or not.

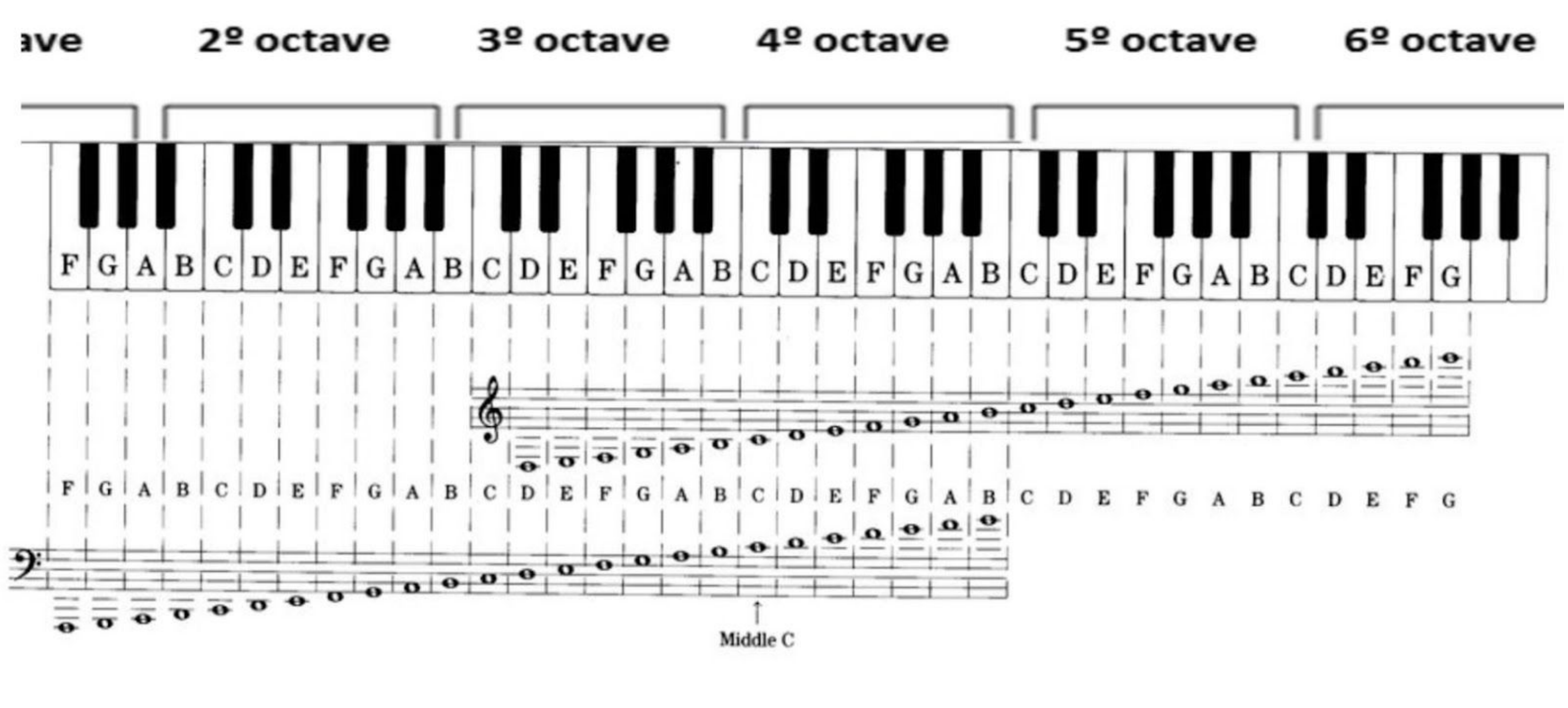


        2. Next to the music notes: **only lower the pitch of the notes in that measure** by one semitone. A measure is one of the spaces between the vertical lines. The example below shows the pitch **D5**. If the flat symbol is right beside it, you need to lower only it to **DF5**.

        3. In the image above, you can see three flats next to the clef on the B, E, and A pitches. Every B, E, and A note in this music will be flat, regardless of pitch (i.e., BF3 BF4 BF5, EF3 EF4 EF5, AF3 AF4 AF5...) unless a natural sign (♮, see below) is used to remove the flat. The D5 is also flat, but only for that measure.

- **Natural (♮)**: cancels a flat or a sharp
  - The natural symbol is used when a flat or sharp is put next to the clef but you want to negate that flat or sharp for one note or measure.

When there is a flat or sharp next to the clef, you would typically change all the notes specified. With a **natural,** it is specifically **not changed**. In other words, it cancels the flat or sharp. For example, if the clef had a sharp next to it on the pitch F, this specific pitch will remain an **F4** instead of getting converted to an FF4.



Pitch to Notation Table

3.2 Decipher Note Duration

Now that you have an array of pitches for each note, we need to find the duration to complete each note in our composition. In the Durations column, list your durations so they line up with the correct pitch in the Pitches column.

NOTE: Sharp, flat, and natural symbols do not have durations. They only change the pitch, not the duration.

Use the *Lengths of Notes (Durations)* Table below to convert the duration of a pitch into notation

Name	Codes in Arduino	Beats	Note	Rest
Whole Note	WHOLE	4		
Dotted Half Note	DOTTED_HALF	3		
Half Note	HALF	2		
Dotted Quarter Note	DOTTED_QUARTER	11/2		
Quarter Note	QUARTER	1		
Dotted 8th Note	DOTTED_EIGHTH	3/4		
8th Note	EIGHTH	1/2		
Dotted 16th Note	DOTTED_SIXTEENTH	3/8		
16th Note	SIXTEENTH	1/4		
32nd Note	THIRTY_SECOND	1/8		

Lengths of Notes (Durations)

	These are two eighth notes. Two or more eighth notes are usually connected and written this way.
	These are four sixteenth notes. Two or more sixteenth notes are written like this.

EXAMPLE | Convert the Duration into Notation

This pitch has:
1) A black circle
2) A tail
3) A dot

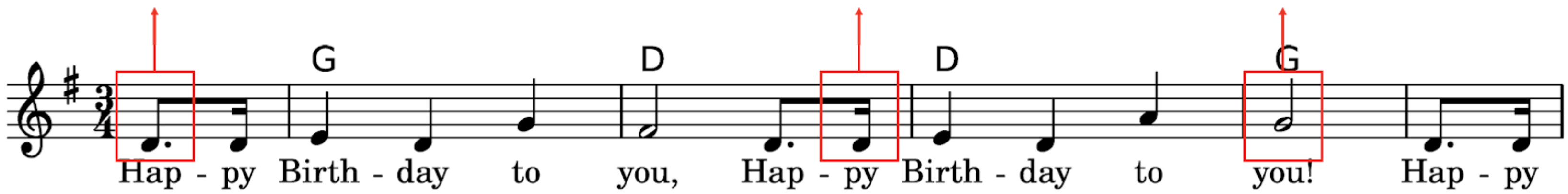
So based on Table 1, the duration of this pitch is DOTTED_EIGHTH

This pitch has:
1) A black circle
2) Two tails
3) No dots

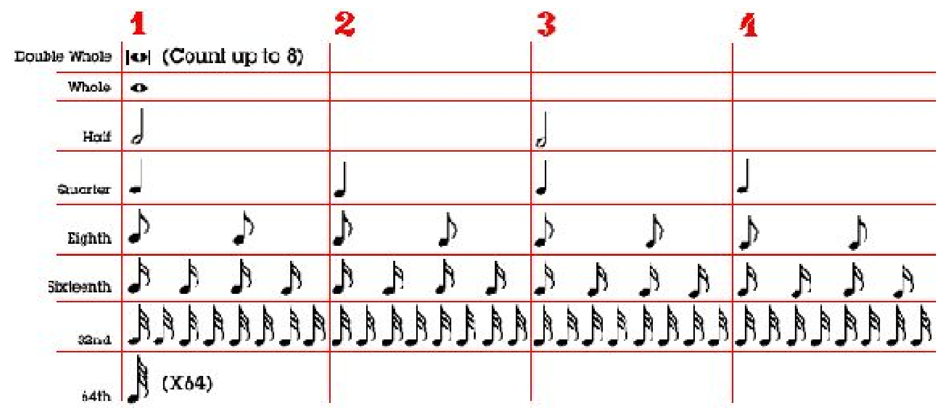
So based on Table 1, the duration of this pitch is SIXTEENTH

This pitch has:
1) A hollow circle
2) No tails or dots

So based on Table 1, the duration of this pitch is HALF



Do you wonder how the durations of notes got their names? It's all fractions!



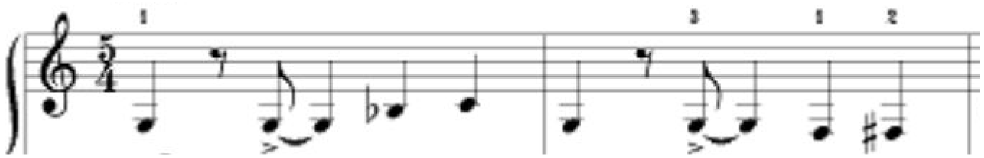
Whole note: A whole note is the longest note, similar to a whole number. A whole note is equivalent to "1."
Half note: A half note is exactly that—*half* of a whole note, ½.
Quarter note: A quarter note is one-fourth (¼) of a whole note. Just like quarters are in a dollar.
Eighth note: An eighth note is an eighth of the duration of a whole note, ⅛.
Sixteenth note: A sixteenth note is a sixteenth as long as a whole note, 1/16th.

3.3 Code Your Music

You have a column of pitches, and a column of durations. Now you need to translate them into hertz and decimal fractions, and enter them into arrays.

Format: `MyPitches[] = {pitch1, pitch2,...};`
`MyDurations[]={duration1, duration2,...};`

Example:



MyPitches[]={G3, 0, G3, BF3, C4, G3, 0, G3, F3, FS3 };
MyDurations[]={quarter, eighth, dotted_eighth, quarter, quarter, quarter, eighth, dotted_eighth, quarter, quarter};

NOTE: See the two sets of 'smiley face' notes with the line connecting them beneath the notes? The 'smile' that connects the two notes tells us to combine the duration of both notes (equal to ⅛ plus ¼ = ⅜) and only play the note **once**, just holding the note longer than you normally would. To write it in our array it is MyPitch[G3] and MyDuration[dotted_eighth].

Pitches Table

SILENCE	0	C2	65	C3	131	C4	262	C5	523	C6	1047	C7	2093
B0	31	CS2	69	CS3	139	CS4	277	CS5	554	CS6	1109	CS7	2217
C1	33	DF2	69	DF2	139	DF4	277	DF5	554	DF6	1109	DF7	2217
CS1	35	D2	73	D3	147	D4	294	D5	587	D6	1175	D7	2349
DF1	35	DS2	78	DS3	156	DS4	311	DS5	622	DS6	1245	DS7	2489
D1	37	EF2	78	EF3	156	EF4	311	EF5	622	DF6	1245	E7	2637
DS1	39	E2	82	E3	165	E4	330	E5	659	E6	1319	F7	2794
EF1	39	F2	87	F3	175	F4	349	F5	698	F6	1397	FS7	2960
E1	41	FS2	93	FS3	185	FS4	370	FS5	740	FS6	1480	G7	3136
F1	44	GF2	93	GF3	185	GF4	370	GF5	740	GF6	1480	GS7	3322
FS1	46	G2	98	G3	196	G4	392	G5	784	G6	1568	A7	3520
GF1	46	GS2	104	GS3	208	GS4	415	GS5	831	GS6	1661	AS7	3729
G1	49	AF2	104	AF3	208	AF4	415	AF5	831	AF6	1661	B7	3951
GS1	52	A2	110	A3	220	A4	440	A5	880	A6	1760	C8	4186
AF1	52	AS2	117	AS3	233	AS4	466	AS5	932	AS6	1865	CS8	4435
A1	55	BF2	117	BF3	233	BF4	466	BF5	932	BF6	1865	D8	4699
AS1	58	B2	123	B3	247	B4	494	B5	988	B6	1976	DS8	4978
BF1	58												
B1	62												

Durations Table

WHOLE	1	EIGHTH_TRIPLE	1.0/12.0	DOTTED_HALF	3.0/4.0
HALF	1.0/2.0	SIXTEENTH	1.0/16.0	DOTTED_QUARTER	3.0/8.0
QUARTER	1.0/4.0	THIRTY_SECOND	1.0/32.0	DOTTED_EIGHTH	3.0/16.0
EIGHTH	1.0/8.0	DOTTED_WHOLE	3.0/2.0	DOTTED_SIXTEENTH	3.0/32.0

EXAMPLE | Converted Sheet Music Code (Melody 3)

Music Code to Start With (found at http://www.exploringcs.org/e-textiles/modules/supporting-code)

```
//Row Your Boat
……………………………………………………………………………………………..
int speaker = 5;
int pace = 2200;
//2200 is the length of a whole note.
//We can increase or decrease this to make the whole song slower or faster
//We are using float for this array instead of int, since int is for whole numbers
//We are doing math right in the array addresses and will get decimal point numbers back!
float MyDurations[] = {3.0/8.0, 3.0/8.0, 1.0/4.0, 1.0/8.0, 3.0/8.0, 1.0/4.0, 1.0/8.0,
1.0/4.0, 1.0/8.0, 3.0/4.0};
int MyPitches[] = {294, 294, 294, 330, 370, 370, 330, 370, 392, 440};


void setup(){
pinMode(speaker, OUTPUT);
}

void loop(){
   for (int i = 0; i < 10; i = i+1) {
      tone (speaker, MyPitches[i]);
      delay(pace * MyDurations[i]);
      notone(speaker);
      delay(10);
   }
}
……………………………………………………………………………………………..
```

EXAMPLE | Translate a 10-Note Melody into Arrays

Notes	Durations
D4	DOTTED_QUARTER
D4	DOTTED_QUARTER
D4	QUARTER
E4	EIGHTH
FS4	DOTTED_QUARTER
FS4	QUARTER
E4	EIGHTH
FS4	QUARTER
G4	EIGHTH
A4	DOTTED_HALF

Changes into…

`MyPitches[]`	`MyDurations[]`
294	3.0/8.0
294	3.0/8.0
294	1.0/4.0
330	1.0/8.0
370	3.0/8.0
370	1.0/4.0
330	1.0/8.0
370	1.0/4.0
392	1.0/8.0
440	3.0/4.0

Which looks like…

```
int MyPitches[] = {294, 294, 294, 330, 370, 370, 330, 370, 392, 440};
float MyDurations[] = {3.0/8.0, 3.0/8.0, 1.0/4.0, 1.0/8.0, 3.0/8.0, 1.0/4.0, 1.0/8.0, 1.0/4.0,
1.0/8.0, 3.0/4.0};
```

Mystery Melodies:
1. Star Wars theme, 2. Harry Potter theme, 3. Row Your Boat, 4. Let It Go, and 5. Mission Impossible