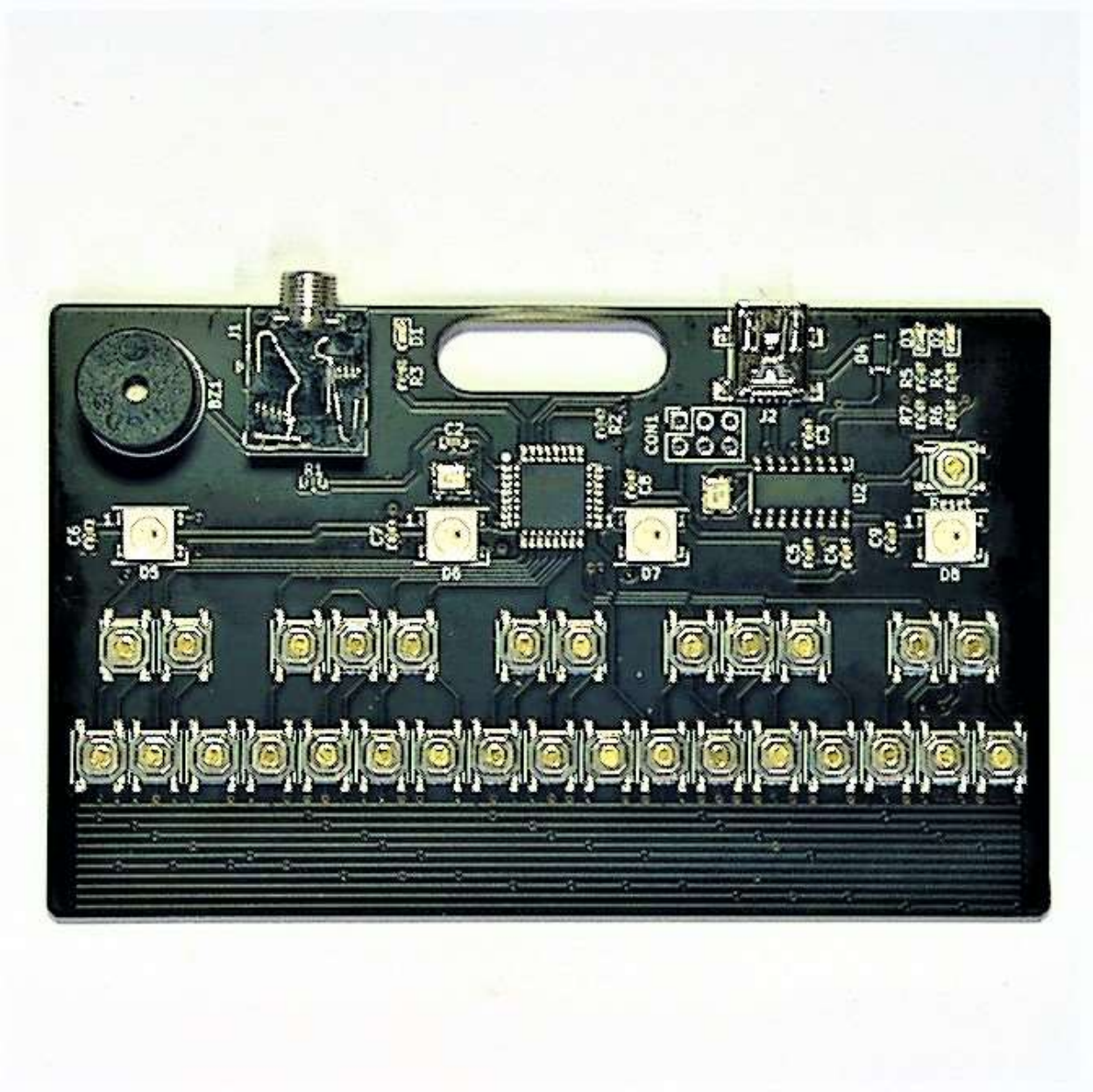


Cyber City Circuits Mini-Tone Keyboard Instructions

David Ray – Draft 20200525

Contents

1. Overview

The Cyber City Circuits Mini Tone Keyboard is an Arduino compatible music keyboard with 2.5 octaves, a *luxurious* **ATMEGA328PB micro-controller**, a *beautiful* **CH-340G** USB interface, 4 *luminous* **WS2812E** RGB LEDs, a *standard* **on-board speaker**, and a *morphenomenal* **3.5mm jack adapter** so that you can connect it to your **existing sound systems/mixers**. The board is fully compatible with the Arduino IDE if you select 'Arduino Nano' in the 'Board Manager'.

The WS2812E LEDs are the same ones that Adafruit uses for their NeoPixel lights and it uses the Adafruit Neopixel library.

2. Using the Arduino IDE

2.1. Setting up the Arduino IDE

You may need to download the Arduino IDE from the Arduino website. (https://www.arduino.cc/)

The version of the firmware on the Micro-Controller is called the 'Arduino Nano'. It comes with the Arduino Board Manager. To set your Arduino IDE to work with this board, change these settings.

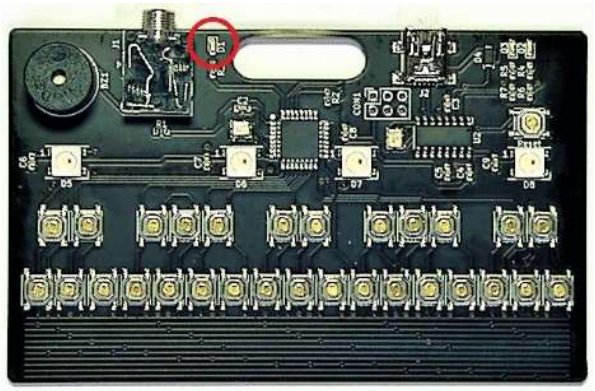
- Tools -> Board -> Arduino Nano
- Tools -> Port -> What Com Port your computer is using for the connection.
    - You may need to check your port number using Windows' Device Manager.

2.2. Loading Your First Program

To make sure you have a good connection to your Mini-Tone Keyboard, we need to send a simple program to it to see if it is communicating. The Mini-Tone Keyboard has a light on the upper left side of the board marked 'D1'. One the micro-controller, that light is connected to pin '2'. Let us write a simple script that will be used to turn that light on and off. This will help us make sure we have a good connection to the board.

Open your Arduino IDE and enter in the following, we will go over what all this means more later.

```
void setup(){
  pinMode(2, OUTPUT);
}
void loop(){
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
  delay(1000);
}
```



Make sure things like ‘digitalWrite’ has a capital ‘W’ in it and ‘pinMode’ has a capital ‘M’ in it. When you’re pretty sure you have it right, you want to ‘push’ your new program to the board.

- Sketch -> Upload

If it worked correctly, you will see the lights on the right side of the board blink on and off a bunch and then the light on the left should blink on for one second and then off for one second. If this is not happening, go back and double check your settings and your program.

3. Introduction to Programming

Now that we have the starter kit built, we can write code for it and make it do things! We are going to be using the Arduino IDE software to program the starter kit.

3.1. Programming Concepts:

There are several basic concepts that we need to go over first. We won’t be going over all of them, but you will be getting a great start. The Arduino uses a type of the C++ programming language. The programs you write for Arduino is called a ‘sketch’.

3.2. Comments

Comments are just that, comments. Comments are ignored by the compiler and do not affect your code at all when used correctly. They are used to make notes and reminders in the code for reference in the future. All good code has a lot of comments in it. There are two types of comments, single line and multiline.

- **Single Line Comments -** Single line comments always start with ‘//’. For example,
  ‘//The LED is connected to digital pin 13.’
  This line tells the reader that there is a LED and it is connected to pin 13 and that pin is a digital pin.
- **Multi Line Comments -** Multi line comments start with ‘/*’ and end with ‘*/’. They are used when you have a lot of information you want to block off. Example:

```
/*
  Blink
  Turns an LED on, then off, repeatedly.
*/
```

3.3. Variables

Variables are used to store values. Here is an example: ‘int a = 13;’. Here you will find three parts of this example.

- **Variable Type** - ‘int’ is used to declare the variable type. ‘Int’ is used to tell your sketch that the variable is an integer. There are a lot of different variable types, but we will primarily use this type in the beginning. An integer can store any number from -32,768 to 32,767. You can’t store any number outside this range as an integer though.

- **Variable Name** – ‘a’ is the name of the variable. A variable name can be any letter from A to Z, a to z, any number from 0 to 9 and the underscore.

- **Variable Value** – In the example, the value we are setting ‘a’ to is ‘13’.

3.4. Functions

Functions are a set of instructions that can be anywhere in a sketch. They are used to perform very specific and repetitive tasks. A function has four major components. Here is an example of a small function.

```
int multi(int a, int b){
    return a * b;
}
```

- **Return Type** – In this example, ‘int’ is called the return type. You might find other things in this spot, like void or float (…and we’ll go over what those mean later…). If you have a function that is used to send a value from that function to another, then you need to tell your ‘sketch’ what type of value that will be. In this example, the value being returned in an integer, so we use ‘int’.

- **Function Name** – The function name in this example is ‘multi’. A function name can be made up of characters from A to Z (lowercase or upper case), numbers from 0-9 and we can use underscores (_). You cannot use the same function name more than once in your sketch. There are also some internal names (…like tone…) that you cannot use.

- **Arguments** – Inside the parenthesis you can place arguments. These are used to give the functions additional information. Even if your function doesn’t need an argument, you must put parenthesis, just leave them empty. In the example there are two arguments, ‘ int a’ and ‘int b’. These are two variables that the function uses to do math with. If you don’t tell the function what numbers to use it won’t work correctly.

- **Statement** – The statement is the actions that the function does when it is called. In the example, the statement is ‘{ return a * b; }’. A statement will always begin and end with curly brackets. This statement will take the number for ‘a’ and the number for ‘b’ and multiply them together. Then, it will ‘return’ the product of the two to the sketch. Almost everything you do, outside of the basic setups, will take place in a function.

3.5. Essential Functions

Every sketch will have two functions to start with. These are ‘void setup()’ and ‘void loop()’.

- The **‘void setup()’** function is an essential part of an Arduino sketch, even if you don’t actually use it. It is the first function read by your program and setups your starter kit to run your sketch. Remember, the setup function only runs once and

at the beginning.

- The **'void loop()**' function is also very essential. It is read after the setup function and will usually contain a very large portion of the operations for the sketch. The function is called loop and it does exactly that – loops. The loop function will run over and over until something tells it to stop. Remember, that the loop function runs over and over again.

3.6. Statements

Statements come in two flavors. Simple and conditional.

- **Simple Statements** - A simple statement is usually a single line and will **always** end with a semi-colon (;). That last part is important. If you have issues with your code, it is most likely a missing semi-colon. An example would be 'digitalWrite(13, HIGH);'.

- **Conditional Statements** – Conditional statements comprise of a condition followed by a series of other statements that only run when the condition is met. Let's walk through another example:

```
if (a = = 5){
    digitalWrite(13, HIGH);
}
else{
    digitalWrite(13, LOW);
}
```

Okay, I know it looks like a lot, but it's pretty simple. Basically when 'a' is equal to 5, then a light will turn on. We will go over it line by line.

Line 1 – This is the conditional statement portion. It reads out as 'if the variable a is equal to 5, then do the following'. You may have noticed that the conditional statement has a double-equals sign (= =) in it. This is different than a single equals sign. The double-equals sign means 'is equal to'. So, the statement after the curly bracket will only execute when 'a' is equal to 5.

Line 2 – This line is a simple statement. digitalWrite is a function name that takes one of the pins of the starter kit and either turns it on or turns it off. When you make a pin 'HIGH' it turns it on, alternatively when you make a pin 'LOW' it turns it off. In this case, it is pin 13 on your starter kit.

Line 3 – This is the closing bracket for the conditional statement we started on line 1.

Line 4 – This is another kind conditional statement. Else is a used with 'if' statements. Basically, when an 'if' condition is not met, then the 'else' statement

is executed. So if 'a' is not equal to 5, then the statements after the curly brackets will be executed.

Line 5 – This is the statement for the 'else' condition. When 'a' is not equal to 5, then we will make the led pin LOW. Remember, this turns the pin off.

Line 6 – This is the closing bracket for the else conditional statement we started on line 4.

3.7. Math and Logic

The Arduino is capable of basic math.

3.7.1. Basic Operators

These are used to do super basic math.

This table shows the difference math operators:	
Operator	**Action**
+	Addition
-	Subtraction
*	Multiplication
/	Division
=	Set Value

These should all pretty straight forward. The one thing to remember is that the microcontroller can only do basic math, so it doesn't always know how to handle division. When dividing two integers, the controller will only return an integer. For example, 7 / 4 would normally give your 1.75, but because it will only return an integer it will give you 1 instead, but if one of the numbers is a float type (decimal number) is will give you a more accurate number. So, if you tried 7.0 / 4 then it will give you 1.75, which is a float instead of an integer.

3.7.2. Compound Operators

These are basically shorthand for math. You can use them to make statements shorter and more concise.

This is the full list of compound operators	
Operator	**Action**
+ +	Increment by 1

- -	Decrement by 1
+ =	Compound addition
- =	Compound subtraction
*** =**	Compound multiplication
/ =	Compound division

Here is an example: 'X = X + 1' - In this example we take whatever value is stored in the 'X' variable and we're just going to add 1 to it. So, if 'X' was 7, after this it will be 8. Instead of writing that, we could just write 'X + +'.

Now we understand how '+ +' and '- -' work, let's talk about how to increment by more than 1. If we use 'X = X + 4', just like before it will add whatever value is stored in X and add 4 to it. We could also use 'X + = 4' and it will do the same thing.

3.7.3. Comparison Operators

Comparison operators are used to compare two different values. We used one of these earlier when we were talking about if statements. The '= =' is a comparison operator.

This is a list of comparison operators:	
Operator	**Action**
= =	Is equal to
! =	Is not equal to
<	Less than
>	Greater than
< =	Less than or equal to
> =	Greater than or equal to

These work just like they sound like. You will mostly uses these in conditional statements, like:

```
if ( a > = 20 ){
                digitalWrite(13, HIGH);
               }
```

In this example, when a is greater than or equal to 20 the pin 13 will turn on.

3.7.4. Boolean Operators

Boolean operators, sometimes called logic operators, are used to test different conditions in if statements. These will return either TRUE or FALSE.

There are three different Boolean operators.		
Operator	**Symbols**	**Action**
AND	&&	True only if both conditions are true
OR	||	True if either condition is true
NOT	!	True if neither condition is true

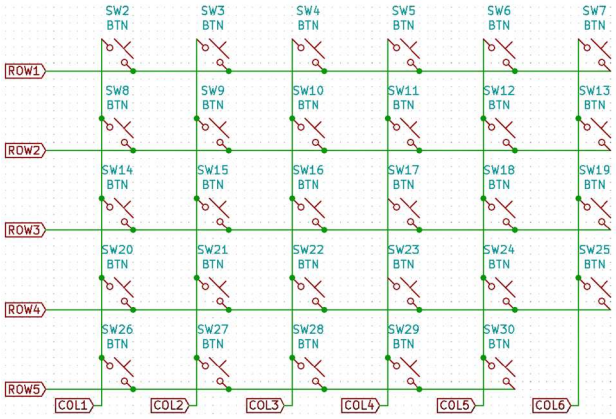
The words AND, OR, and NOT aren't written into the code, but instead you will use symbols. Here is an example:

```
If (x = = 15 && y > = 50){
    digitalWrite(13, HIGH);
}
```

Here the statement, 'digitalWrite(13, HIGH);' is only executed when both 'x' is equal to 14 AND y is greater than or equal to 50.

4. Learning More About Your Mini-Tone Keyboard

4.1. Button Matrixes



The Mini-Tone Keyboard is laid out in a button matrix. Normally we would need to use a single pin for each button, meaning we would need to use 29 digital pins to control the buttons, but by using a matrix we are able to use all 29 buttons with only 11 pins on our micro-controller. To do this, we need to place the buttons in a grid pattern, segmenting the buttons into individual rows and columns. With them broken down in that way, we can just check each row for a button being pressed, then move to the next row, etc.

Normally when we use a button we check the button state with a 'digitalRead()' function. We see if it is LOW or HIGH and then have the boards act accordingly, but in a matrix we can activate one row at a time then check each column of that row. I know this sounds a little complicated, but luckily someone wrote a library called 'Keypad' to do all the work for you.

Each of the row and columns are connected to pins on the micro-controller.

When we write software using those buttons, we will need to know what pins those are connected to.

4.2. The Tone Function

The Arduino IDE has a function built-in called ‘tone’. The ‘tone’ function generates a square wave of the specified frequency (and 50% duty cycle) on specific pin. The pin can be connected to a piezo buzzer or other speaker to play tones and only one tone can be generated at a time. We use this tone function to play music on the Mini-Tone Keyboard. The ‘tone’ function takes three arguments. Arguments are used to pass different settings and information to the function. The arguments for the ‘tone’ function are (1) speaker pin, (2) note to play, (3) duration of the note.

4.3. The RGB Lights

The Mini-Tone Keyboard has four (4) WS-2812E LEDs on it. These LEDs are very popular with people because they are very easy to use. While programming your Mini-Tone Keyboard, we will be using a library from a company named Adafruit. They write all kinds of Arduino software and we will be using the Adafruit Neopixel library to interact with these lights.

5. Introduction to Libraries and Headers

5.1. Libraries

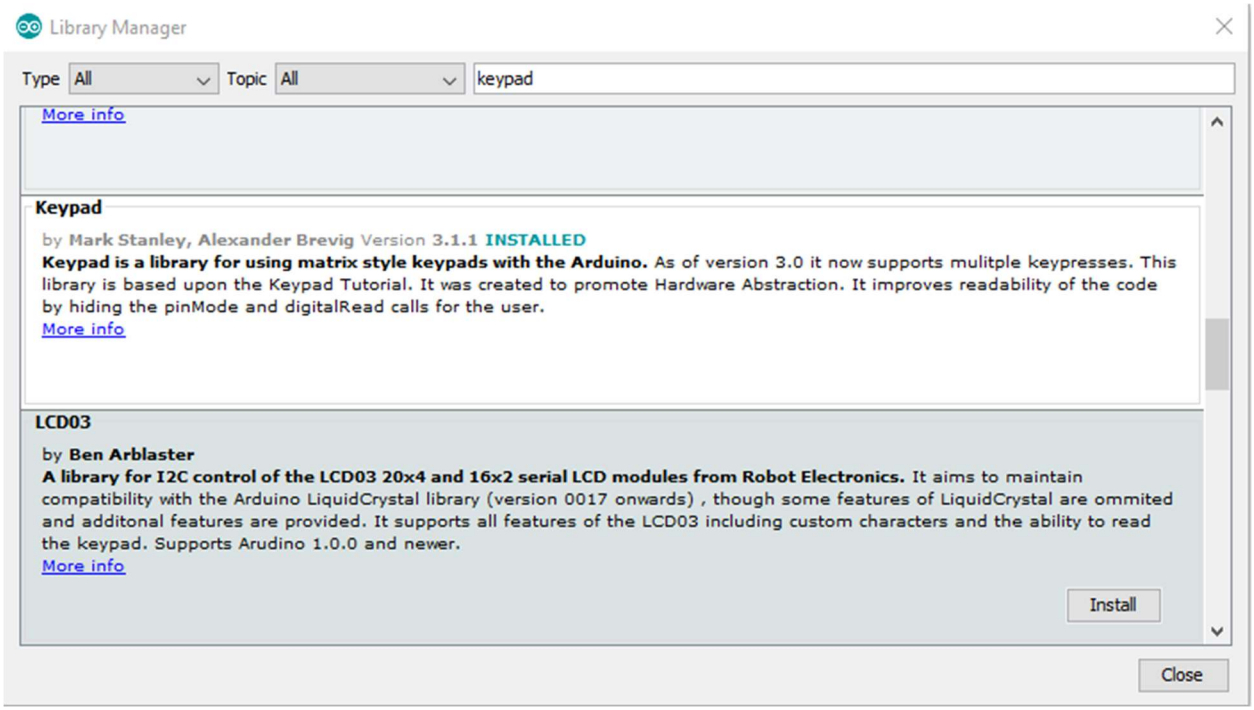
A library is basically a small file or group of files that make it easier to share functionality among different programs. For example, we will need to use a very complex process to use the button matrix. Instead of writing a new process every time we want to use a button matrix we could just use a library. That way we only need to get it working once and then we can easily take that functionality to other projects.

5.2. Using the Library Manager

To open the Arduino Library Manager, within the Arduino IDE click the ‘Tools’ button in the top menu. From there click ‘Manage Libraries…’. This is where we will go to install new libraries to your Arduino IDE. We need two libraries.

5.2.1. Installing the Keypad Library

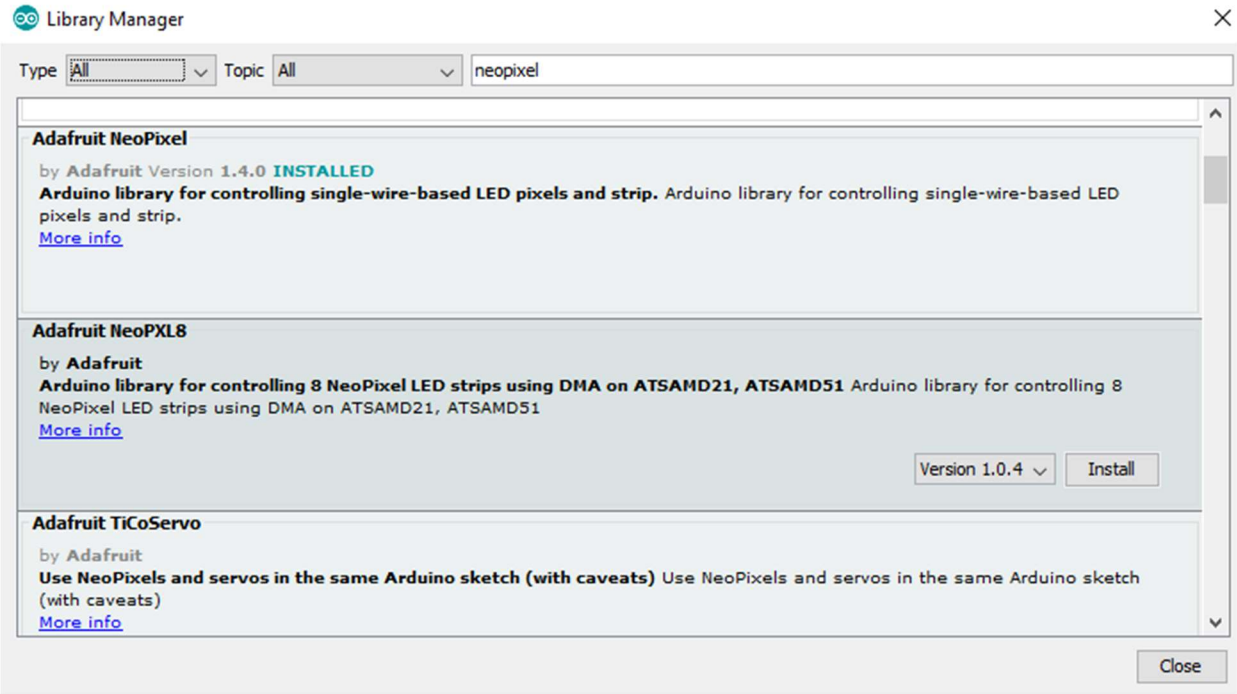
The first library we need to install is the ‘keypad’ library. There are many different ‘keypad’ libraries available, but they are all different and if you install the wrong one your program may not work correctly. On the top of the Library Manager window there is a search bar. Type ‘keypad’ into the search bar and then scroll down to find the library named ‘Keypad’ by Mark Stanley and Alexander Brevig. Once you find it, click install. The version available may be different than what you



see in the example below, but the latest version will work fine.

5.2.2. Installing the Adafruit Neopixel Library

The next library we will need is the 'Adafruit Neopixel' library. In the search bar of the Library Manager, type in 'neopixel'. Scroll down to find the 'Adafruit Neopixel' library by Adafruit and click install.

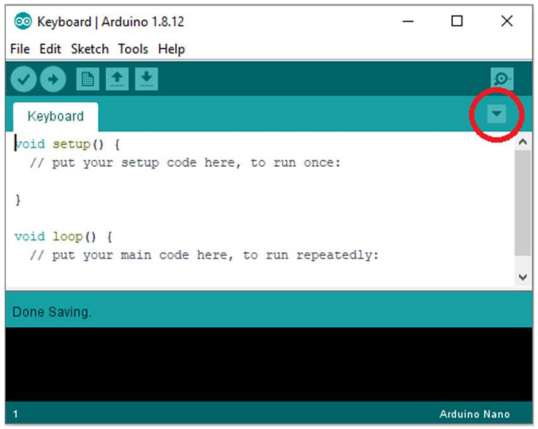


5.3. Header Files

When programming with the Mini-Tone Keyboard, we will be creating a header file named 'pitches.h'. This file will be used to store the note values for the keyboard. Essentially, all it will be is a lookup table for the keyboard to reference when trying to play notes. Writing things like this in a header file will help you keep your code better organized and will make it easier to use the same variables in other pojects. The header file 'pitches.h' will need to be in the same folder as your main keyboard program.

6. Let's Start Writing Code!

Open the Arduino IDE and click the 'File' menu in the top bar and then click 'New'. Next, go ahead and click the 'File' menu again and click 'Save'. A new window will open asking you what you would like to name your program. Name your program 'Keyboard'. Now you have your program saved and we need to add our header file. On the right of your Arduino IDE screen, you will see a little button with a down button arrow. Click on that button and then click on 'New Tab'. Then



on the bottom of the Arduino IDE it will ask you for a name for your new file. We need to name that file 'pitches.h', all lowercase.

6.1. Setting Up Your Header File

As you read before, the 'pitches.h' header file is a way to store the note values. Near the top of your Arduino IDE you will see two tabs. One is named 'Keyboard' and one is named 'pitches.h'. You will need to add the following lines to the 'pitches.h' file. Go ahead and add them into there now.

```
#define C4  262
#define CS4 277
#define D4  294
#define DS4 311
#define E4  330
#define F4  349
#define FS4 370
#define G4  392
#define GS4 415
#define A4  440
#define AS4 466
#define B4  494
#define C5  523
#define CS5 554
#define D5  587
#define DS5 622
#define E5  659
#define F5  698
#define FS5 740
#define G5  784
#define GS5 831
#define A5  880
#define AS5 932
#define B5  988
#define C6  1047
#define CS6 1109
#define D6  1175
#define DS6 1245
#define E6  1319
```

Great! Now you can reference these variables easily from within your main program.

6.2. Playing Mary Had A Little Lamb

Now we are going to create a simple and basic program to play a song on your Mini-Tone Keyboard. Mary Had A Little Lamb is a super simple song and is a great way to learn to use the tone function. Let's set it up step-by-step. You may remember that earlier we mentioned two essential functions. These essential functions must be in every Arduino program for it to operate correctly. They are named 'setup()' and 'loop()'.

In the space before the ‘setup()’ function we will need to add a few lines of code.

The first line will tell the program to use the ‘pitches.h’ header file that we just made.

```
#include "pitches.h"
```

After that we need to tell the Arduino IDE what pin our speaker is on. To do this we will create a variable named ‘speaker’ and make that equal to the pin that the speaker is connected to on the micro-controller.

```
int speaker =  3;
```

The notes we use for Mary Had A Little Lamb are:
‘B5, A5, G5, A5, B5, B5, B5, A5, A5, A5, B5, B5, B5’

To play a note we will use the ‘tone’ function followed by the ‘delay’ function. We have to use the delay function so that the notes that we play with the ‘tone’ function don’t blend together. Go ahead and type in the following ‘tone’ and ‘delay’ functions into your ‘loop()’ function.

```
void loop(){
    tone(speaker, B5, 250);
    delay(500);
    tone(speaker, A5, 250);
    delay(500);
    tone(speaker, G5, 250);
    delay(500);
    tone(speaker, A5, 250);
    delay(500);
    tone(speaker, B5, 250);
    delay(500);
    tone(speaker, B5, 250);
    delay(500);
    tone(speaker, B5, 250);
    delay(500);
    tone(speaker, A5, 250);
    delay(500);
    tone(speaker, A5, 250);
    delay(500);
    tone(speaker, A5, 250);
    delay(500);
    tone(speaker, B5, 250);
    delay(500);
    tone(speaker, B5, 250);
    delay(500);
    tone(speaker, B5, 250);
    delay(500);
}
```

and then upload your sketch. If everything worked, you should hear that magical

square-tone lullaby of the Mary Had A Little Lamb on your Mini-Tone Keyboard.

Note: You will probably want to disconnect the board from the USB so the music will stop. You can plug it back in later when you are ready to push your new code.

6.3. Introducing Variables into the Code

That was very tedious wasn't it? What if you wanted to make the delays last a little longer or make them shorter? You would have to change every single instance of delay in your code and that can be exhausting if you do it enough. This is where variables can come in real handy. In the area before the 'setup()' function, add a variable named 'wait' and make it equal to 300.

```
int wait = 300;
```

Now we want to change every 'delay' function to use this new 'wait' variable. To do this, we are going to use the find and replace tool in the Arduino IDE. Click on the 'Edit' menu in the menu bar and then click on 'Find…'. A new window will pop up with two fields in it. Put 'delay(500);' into the 'Find:' field and then put 'delay(wait);' into the 'Replace with:' field. Then click the 'Replace All' button. Then, like magic, all the 'delay' functions have changed.

Go ahead and upload the sketch to your Mini-Tone Keyboard. You should hear the same notes being played as before, but the space between the notes is a lot shorter and the music should sound faster. Good job!

6.4. Creating a Function to Make Your Code Cleaner

Okay, the program is starting to get a little easier to work with, but it is still very tedious to use. This is where creating your own functions can make it a lot easier to work with your code.

In this exercise we are going to create a function called 'playNote'. We want to be able to play a note and turn the light 'D1' on when a note is playing. Before we can use the light 'D1' we have to ass a couple of lines to our code. First, we have to assign the light to a pin number. In the area before the 'setup()' function, add the following line:

```
int led = 2;
```

Then, we need to tell our Arduino IDE that we need to make this light an OUTPUT. Inside the 'setup()' function add the following line:

```
pinMode(led, OUTPUT);
```

Now, we can starting using the light 'D1' with our code!

In the last session we made the wait variable to change how long the controller waited before playing the next note. Let's add another variable for how long the note plays. In the area where you define the wait variable, make a play variable and in the code where it says 250 for the play length, we will use the play variable in the future.

```
int play = 250;
```

Now to add the function to our code. So far all of the code has been within the 'setup()' and 'loop()' functions. To create a new function, you need to add them AFTER the 'loop()' function. We want our 'playNote' function to do a few things. We want it to

(1) play a note, we want to be able to (2) tell the code how long the note will be, and then (3) tell the code how long to rest before playing the next note. So, we will need to add those three arguments into the function. Then once the function is called, we want (4) the light 'D1' to turn on, (5) play the note, when the note is done playing we want (6) the light 'D1' to turn off, then we want it the function to (7) wait to play the next note. Add the following lines of code in the area AFTER the 'loop()' function.

```
void playNote(int note, int duration, int rest) {
  digitalWrite(led, HIGH);
  tone(speaker, note, duration);
  digitalWrite(led, LOW);
  delay(rest);
}
```

Now, instead of playing the note and having it wait in two lines over and over, we can just call this function. Below is an example of how we replaced four lines that we would need to write over and over again, with just a single line by using the function we just wrote.

Before	After

`digitalWrite(led, HIGH);`
`tone(speaker, A5, play);`
`digitalWrite(led, LOW);`
`delay(wait);`
	`playNote(A5, play, wait);`

The next step is going to be to change out all the lines in the 'loop()' function to use the new 'playNote()' function. Here is a reminder that every statement needs to end with a semicolon (';') and the 'loop()' function has to have a closing curly bracket at the end ('}').

If this was done correctly you should not notice any difference at all from how it sounds.

6.5. Creating the Button Matrix Within Your Script

Earlier in this document you read about the button matrix. Essentially it is a lot of buttons laid out in rows and columns to allow you to use a lot more buttons than you have pins for. The Mini-Tone Keyboard has 29 buttons, but it only uses eleven pins on the micro-controller to accomplish this. Five pins are used for the rows and six pins are used for the columns. I'm going to list those pins now for reference.

Variable	Pin Used	Variable	Pin Used
row_1	6	col_1	8
row_2	10	col_2	5
row_3	11	col_3	15
row_4	14	col_4	7
row_5	13	col_5	9
		col_6	12

You may notice that the pin numbers aren't in any particular order, this is because when you are designing a PCB, sometimes you have to connect the rows and columns anyway you can to make a clean board.

Now that we understand a little better how the physical buttons are laid out, we need to write it out in a way that the software we write will be able to use it. Earlier you downloaded the 'Keypad' library, let's walk through how that works a little. The 'Keypad' library is used to create a keypad object in your code.

```
Keypad buttons = Keypad(makeKeymap(keys), row[], col[], rows, cols);
```

That is what the syntax for creating a Keypad object named 'buttons' looks like. Let's take a moment and walk through this step-by-step.

`Keypad buttons`	Calls the Keypad object type created by the library and creates an object named 'buttons'.
`Keypad(`	Calls a 'Keypad' function with the library.
`makeKeymap(keys)`	Creates a map for the buttons with names.
`row[]`	List of pins used for rows.
`col[]`	List of pins used for columns.
`Rows`	Count of how many rows there are.
`Cols`	Count of how many columns there are.

Because the button matrix is a grid of five rows and six columns, we need to create variables in our program file to tell the 'Keypad' library how to read our button matrix.

```
const byte ROWS = 5; //There are three rows
const byte COLS = 6; //The are six columns

//Assign names to the buttons
char keys[ROWS][COLS] = {
{'1','2','3','4','5','6'},
{'7','8','9','a','b','c'},
{'d','e','f','g','h','i'},
{'j','k','l','m','n','o'},
{'p','q','r','s','t','u'}
};

//connect to the row pinouts of the keypad
byte rowPins[ROWS] = {row_1, row_2, row_3, row_4, row_5};
//connect to the column pinouts of the keypad
byte colPins[COLS] = {col_1, col_2, col_3, col_4, col_5, col_6};

//Construct the Keypad object called 'buttons'
Keypad buttons = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
```

Using the table above, where I describe the arguments needed to create a 'Keypad' object, you should be able tell how the variables for the arguments are created and passed into the construction of the Keypad object. So, later we can call a function called 'buttons.getKey()' and the Keypad object will check and see what button is being pressed and return the name that is in the keymap named 'keys'.

Next, we need to create an array of notes that will be played. We're going to build this array so that when a button is pressed, the software can just an associated note value.

```
//An array of note values to associate to the array of buttons (keys)
int notes[29] = {
    C4, CS4,  D4, DS4,  E4,  F4,
   FS4,  G4, GS4,  A4, AS4,  B4,
    C5, CS5,  D5, DS5,  E5,  F5,
   FS5,  G5, GS5,  A5, AS5,  B5,
    C6, CS6,  D6, DS6,  E6
};
```

That's all we need to do to actually setup the button matrix. Now, we have to write the code to use the button matrix. We need to create another function that is going to be used to read the Keypad object and then play a note. We will name the function 'check_keyboard()'.

Inside the 'check_keyboard()' function we will read the keypad object with the following statement.

```
char key = buttons.getKey();
```

This statement checks the Keypad object named buttons and returns the name of the button being pressed into a variable named 'key'. The variable 'key' will only have a value in it when a button is recognized. If we want to check to see if a value is stored in 'key' we can do that simply by using an 'if' statement of 'if (key){}'. When there is a value stored in key, it will pass the 'if' statement and then we can compare the value of 'key'.

So, when a key is pressed and if the key value is '1' (the first button on the keymap.), then we want our code to play the first note in the 'notes' array. Then, if the key value is '2', then we want to play the second note in the 'notes' array, and so on. The code should look something like the following.

```
void check_keyboard(){
  char key = buttons.getKey();

  //Find the value of the key press and play a note.
  if (key){
    if (key == '1'){playNote(notes[0],  play, wait);}
    if (key == '2'){playNote(notes[1],  play, wait);}
    if (key == '3'){playNote(notes[2],  play, wait);}
```

```
    if (key == '4'){playNote(notes[3],  play, wait);}
    if (key == '5'){playNote(notes[4],  play, wait);}
    if (key == '6'){playNote(notes[5],  play, wait);}
    if (key == '7'){playNote(notes[6],  play, wait);}
    if (key == '8'){playNote(notes[7],  play, wait);}
    if (key == '9'){playNote(notes[8],  play, wait);}
    if (key == 'a'){playNote(notes[9],  play, wait);}
    if (key == 'b'){playNote(notes[10], play, wait);}
    if (key == 'c'){playNote(notes[11], play, wait);}
    if (key == 'd'){playNote(notes[12], play, wait);}
    if (key == 'e'){playNote(notes[13], play, wait);}
    if (key == 'f'){playNote(notes[14], play, wait);}
    if (key == 'g'){playNote(notes[15], play, wait);}
    if (key == 'h'){playNote(notes[16], play, wait);}
    if (key == 'i'){playNote(notes[17], play, wait);}
    if (key == 'j'){playNote(notes[18], play, wait);}
    if (key == 'k'){playNote(notes[19], play, wait);}
    if (key == 'l'){playNote(notes[20], play, wait);}
    if (key == 'm'){playNote(notes[21], play, wait);}
    if (key == 'n'){playNote(notes[22], play, wait);}
    if (key == 'o'){playNote(notes[23], play, wait);}
    if (key == 'p'){playNote(notes[24], play, wait);}
    if (key == 'q'){playNote(notes[25], play, wait);}
    if (key == 'r'){playNote(notes[26], play, wait);}
    if (key == 's'){playNote(notes[27], play, wait);}
    if (key == 't'){playNote(notes[28], play, wait);}
  }
}
```

Finally, we need to add the ‘check_keyboard()’ function into the ‘loop()’ function so that the keyboard will continuously check the button matrix. Upload your script and play with the buttons. It should work great!

7. Appendix – Code References

7.1. Example – 2.2 – Blink

```
void setup(){
  pinMode(2, OUTPUT);
}
void loop(){
  digitalWrite(2, HIGH);
  delay(1000);
  digitalWrite(2, LOW);
 delay(1000);
}
```

7.2. Example – 6.1 – Setting Up Your Header File

```
#define C4  262
#define CS4 277
#define D4  294
#define DS4 311
#define E4  330
#define F4  349
#define FS4 370
#define G4  392
#define GS4 415
#define A4  440
#define AS4 466
#define B4  494
#define C5  523
#define CS5 554
#define D5  587
#define DS5 622
#define E5  659
#define F5  698
#define FS5 740
#define G5  784
#define GS5 831
#define A5  880
#define AS5 932
#define B5  988
#define C6  1047
#define CS6 1109
#define D6  1175
#define DS6 1245
#define E6  1319
```

7.3. Example 6.2 – Playing Mary Had A Little Lamb

```
#include "pitches.h"

int speaker =  3;

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  tone(speaker, B5, 250);
  delay(500);
  tone(speaker, A5, 250);
  delay(500);
  tone(speaker, G5, 250);
  delay(500);
  tone(speaker, A5, 250);
  delay(500);
  tone(speaker, B5, 250);
  delay(500);
  tone(speaker, B5, 250);
  delay(500);
  tone(speaker, B5, 250);
  delay(500);
  tone(speaker, A5, 250);
  delay(500);
  tone(speaker, A5, 250);
  delay(500);
  tone(speaker, A5, 250);
  delay(500);
  tone(speaker, B5, 250);
  delay(500);
  tone(speaker, B5, 250);
  delay(500);
  tone(speaker, B5, 250);
  delay(500);
}
```

7.4. Example – 6.3 – Introducing Variables into the Code

```
#include "pitches.h"

int speaker =  3;
int wait = 300;

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  tone(speaker, B5, 250);
  delay(wait);
  tone(speaker, A5, 250);
  delay(wait);
  tone(speaker, G5, 250);
  delay(wait);
  tone(speaker, A5, 250);
  delay(wait);
  tone(speaker, B5, 250);
  delay(wait);
  tone(speaker, B5, 250);
  delay(wait);
  tone(speaker, B5, 250);
  delay(wait);
  tone(speaker, A5, 250);
  delay(wait);
  tone(speaker, A5, 250);
  delay(wait);
  tone(speaker, A5, 250);
  delay(wait);
  tone(speaker, B5, 250);
  delay(wait);
  tone(speaker, B5, 250);
  delay(wait);
  tone(speaker, B5, 250);
  delay(wait);
}
```

7.5. Example – 6.4 – Creating a Function to Make Your Code Cleaner

```
#include "pitches.h"

int speaker =  3;
int led = 2;
int play = 250;
int wait = 500;

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

void loop() {
  playNote(B5, play, wait);
  playNote(A5, play, wait);
  playNote(G5, play, wait);
  playNote(A5, play, wait);
  playNote(B5, play, wait);
  playNote(B5, play, wait);
  playNote(B5, play, wait);
  playNote(A5, play, wait);
  playNote(A5, play, wait);
  playNote(A5, play, wait);
  playNote(B5, play, wait);
  playNote(B5, play, wait);
  playNote(B5, play, wait);
}

void playNote(int note, int duration, int rest) {
  digitalWrite(led, HIGH);
  tone(speaker, note, duration);
  digitalWrite(led, LOW);
  delay(rest);
}
```

7.6. Example – 6.5 - Creating the Button Matrix Within Your Script

```
#include "pitches.h"  //Use the user created library header for note values
#include <Keypad.h>   //Import the 'keypad' library to use the buttom matrix

int play = 250;  //How long to play a note.
int wait = 100;  //How long to rest between notes

//Assign pins
int speaker =  3;  //This is the pin that the speaker is on
int led     =  2;  //This is the pin for the LED that is on board

int row_1 =  6;  //Row 1 of the keyboard matrix
int row_2 = 10;  //Row 2 of the keyboard matrix
int row_3 = 11;  //Row 3 of the keyboard matrix
int row_4 = 14;  //Row 4 of the keyboard matrix
int row_5 = 13;  //Row 5 of the keyboard matrix

int col_1 =  8;  //Column 1 of the keyboard matrix
int col_2 =  5;  //Column 2 of the keyboard matrix
int col_3 = 15;  //Column 3 of the keyboard matrix
int col_4 =  7;  //Column 4 of the keyboard matrix
int col_5 =  9;  //Column 5 of the keyboard matrix
int col_6 = 12;  //Column 6 of the keyboard matrix

const byte ROWS = 5; //There are three rows
const byte COLS = 6; //The are six columns

//Assign names to the buttons
char keys[ROWS][COLS] = {
{'1','2','3','4','5','6'},
{'7','8','9','a','b','c'},
{'d','e','f','g','h','i'},
{'j','k','l','m','n','o'},
{'p','q','r','s','t','u'}
};

//connect to the row pinouts of the keypad
byte rowPins[ROWS] = {row_1, row_2, row_3, row_4, row_5};
//connect to the column pinouts of the keypad
byte colPins[COLS] = {col_1, col_2, col_3, col_4, col_5, col_6};

//Construct the Keypad object called 'buttons'
Keypad buttons = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

//An array of note values to associate to the array of buttons (keys)
```

```
int notes[29] = {
    C4, CS4,  D4, DS4,  E4,  F4,
   FS4,  G4, GS4,  A4, AS4,  B4,
    C5, CS5,  D5, DS5,  E5,  F5,
   FS5,  G5, GS5,  A5, AS5,  B5,
    C6, CS6,  D6, DS6,  E6
};

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

void loop() {
  check_keyboard();
}

void check_keyboard(){
  char key = buttons.getKey();

  //Find the value of the key press and play a note.
  if (key){
    if (key == '1'){playNote(notes[0],  play, wait);}
    if (key == '2'){playNote(notes[1],  play, wait);}
    if (key == '3'){playNote(notes[2],  play, wait);}
    if (key == '4'){playNote(notes[3],  play, wait);}
    if (key == '5'){playNote(notes[4],  play, wait);}
    if (key == '6'){playNote(notes[5],  play, wait);}
    if (key == '7'){playNote(notes[6],  play, wait);}
    if (key == '8'){playNote(notes[7],  play, wait);}
    if (key == '9'){playNote(notes[8],  play, wait);}
    if (key == 'a'){playNote(notes[9],  play, wait);}
    if (key == 'b'){playNote(notes[10], play, wait);}
    if (key == 'c'){playNote(notes[11], play, wait);}
    if (key == 'd'){playNote(notes[12], play, wait);}
    if (key == 'e'){playNote(notes[13], play, wait);}
    if (key == 'f'){playNote(notes[14], play, wait);}
    if (key == 'g'){playNote(notes[15], play, wait);}
    if (key == 'h'){playNote(notes[16], play, wait);}
    if (key == 'i'){playNote(notes[17], play, wait);}
    if (key == 'j'){playNote(notes[18], play, wait);}
    if (key == 'k'){playNote(notes[19], play, wait);}
    if (key == 'l'){playNote(notes[20], play, wait);}
    if (key == 'm'){playNote(notes[21], play, wait);}
    if (key == 'n'){playNote(notes[22], play, wait);}
    if (key == 'o'){playNote(notes[23], play, wait);}
```

```
    if (key == 'p'){playNote(notes[24], play, wait);}
    if (key == 'q'){playNote(notes[25], play, wait);}
    if (key == 'r'){playNote(notes[26], play, wait);}
    if (key == 's'){playNote(notes[27], play, wait);}
    if (key == 't'){playNote(notes[28], play, wait);}
  }
}

//Function to play notes
void playNote(int note, int duration, int rest) {
  digitalWrite(led, HIGH);
  tone(speaker, note, duration);
  delay(rest);
  digitalWrite(led, LOW);
}
```