Topic:
5. Digital oscillator

Summary session:

This session will introduce the fifth lab of the module. The fifth lab of the module is about digital oscillators on Arduino. In this lab, you will learn how to generate melodies, build a digital oscillator with interactive controls and start exploring the basic examples provided by the Mozzi library.

## WARM-UP ACTIVITY

Time: 5’

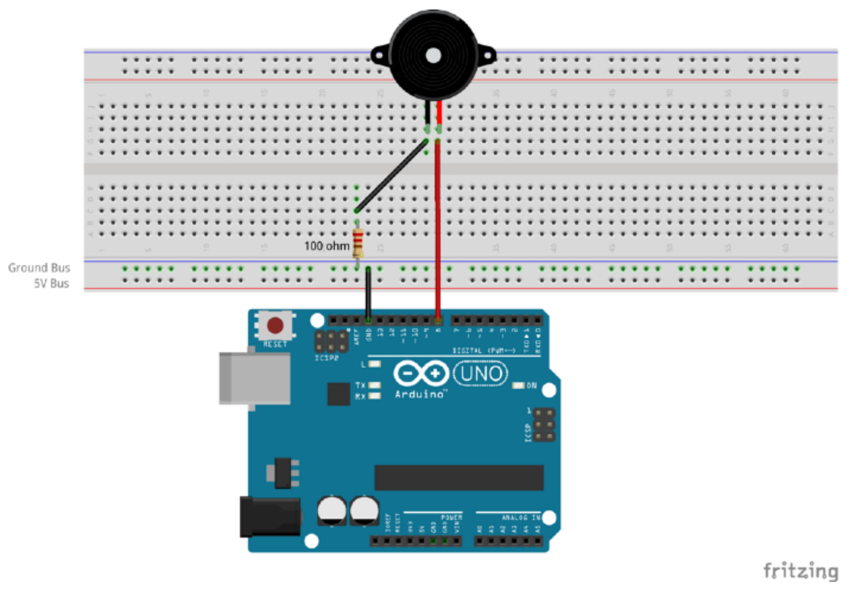
1. Revision of last week’s blog posts.

## BLOGGING

Time: 5’ (setup), 110’ (full activity)

1. **Write your fourth blog post at https://matd3039.our.dmu.ac.uk/**
    1. A good practice is to write a blog post during the whole lab session. This way, you can keep notes of the whole process, which can be useful for later.
    2. Start your fourth blog post of the module.
    3. Make sure to keep writing from the beginning to end of the lab session with key ideas and findings.

## MAKE NOISE USING TONE() AND A PIEZO BUZZER



As you will see in the examples, it is recommended to add a series resistor to the piezo buzzers. The reason is threefold: 1) you make sure that there is a resistor that limits the current to something below 40mA (current of a pin on the Arduino board) and the 5V (voltage provided by the Arduino board); and 2) it reduces the extreme treble response by about 3 dB at 20 KHz, and about 2 dB at 16 KHz; 3) to protect the IC from back electromotive force generated by a piezoelectric sound component. You can find further info here:

* Buzzer on Arduino UNO: https://electronics.stackexchange.com/questions/118090/buzzer-on-arduino-uno
* Please give me an example of the drive circuit for a Piezoelectric Sounder or a Piezoelectric Diaphragm (External Drive Type): https://www.murata.com/en-eu/support/faqs/sound/sounder/char/sch0001 Making piezo tweeters sound better!: http://donklipstein.com/pzfix.html

On a side note, the reason of adding a series resistor to an LED is different, see the explanation here:
* Using Resistors in Arduino (case: limit current for LEDs): https://www.dummies.com/article/technology/computers/hardware/arduino/using-resistors-in-arduino-166942

If you plan to work with piezo buzzers, there are several aspects to consider. Here's a summary: https://www.rs-online.com/designspark/10-things-to-consider-when-purchasing-buzzers-and-sounders

Time: 30'

1. Build the following circuit with the following parts:
   1. Solderless breadboard.
   2. Jumper wires.
   3. 100 ohm resistor.
   4. Piezo speaker (aka piezo buzzer).

2. The connections need to be connected as follows (unplug the USB cord from your Arduino Uno so it is not powered while you make your connections):
   1. Place the piezo buzzer into the breadboard, so that the two leads are on two separate rows.
   2. Using jumper wires, connect the positive lead to Arduino digital pin 8. The case of the buzzer may have a positive sign (+) on it to indicate the positive lead (if not, then the red wire usually indicates the positive lead).
   3. Connect the other lead to the 100 ohm resistor, and then to ground.

3. Write the following code on the Arduino IDE software. Save it as `"TonePiezoBuzzer"`.

```
//A sketch to demonstrate the tone() function

//Specify digital pin on the Arduino that the positive lead of
piezo buzzer is attached.
const int piezoPin = 8;

void setup() {

}//close setup

void loop() {

  /*Tone needs 2 arguments, but can take three
    1) Pin#
    2) Frequency - this is in hertz (cycles per second) which
determines the pitch of the noise made
    3) Duration - how long teh tone plays
  */
  tone(piezoPin, 1000, 500);

  //tone(piezoPin, 1000, 500);
```

```
        //delay(1000);
    }
```

A piezo speaker use piezo-electric material to bend a metal diaphragm which makes noise.

As an experiment, try changing the second argument in tone() to 100, 1000, 10000, 650000 and listen to the effect it has on the audio signal.

You will notice that the higher the number, the higher the pitch that is created.

You can find the description of the `tone()` function at: https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/

As described on the reference webpage:

`Tone()` generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to noTone(). The pin can be connected to a piezo buzzer or other speaker to play tones.

Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to tone() will have no effect. If the tone is playing on the same pin, the call will set its frequency.

Use of the tone() function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).

It is not possible to generate tones lower than 31Hz.

If you want to add some space between the noise, you can try adding a `delay(1000)` after the `tone()`, but if you test this out, you will find it doesn't get you anywhere.

This is because the `tone()` function uses one of the built in timers on the Arduino's micro-contoller. `tone()` works independently of the `delay()` function. You can start a tone and do other stuff – while the tone is playing in the background.

However, you can use the third parameter of the `tone()` function to define the duration of the tone in milliseconds.

```
tone( pin number, frequency in hertz, duration in milliseconds);
```

If you want to generate distinct beats, and you want to do this with the `delay()` function, then you need to keep in mind that the `tone()` function uses one of the built in timers on the Arduino board.

Therefore, if you use 500 milliseconds as the third argument in `tone()`, and follow that by a delay of 1000 milliseconds, you will only be creating a "quiet time" of 500 milliseconds.

```
//This code only generates a delay of 500 milliseconds between the
tone

tone( 8, 2000, 500);

delay(1000);
```

Instead of the time being added together – as in 500 millisecond of noise, and then 1000 milliseconds of delay – the delay and the tone start at about the exact same time, so we get 500 millisecond of tone separated by 500 milliseconds of quiet.

4. Try now to make a beat. Write the following code on the Arduino IDE software. Save it as "`TonePiezoBuzzerBeat`".

```
//A sketch to demonstrate the tone() function

//Specify digital pin on the Arduino that the positive lead of
piezo buzzer is attached.
const int piezoPin = 8;

void setup() {

}//close setup

void loop() {

  /*Tone needs 2 arguments, but can take three
    1) Pin#
    2) Frequency - this is in hertz (cycles per second) which
determines the pitch of the noise made
    3) Duration - how long the tone plays
  */
  tone(piezoPin, 450);
  delay(500);
  noTone(piezoPin);
  delay(500);
}
```

5. `noTone()` frees the tone in a specific pin. Let's modify the program and save it as "`TonePiezoBuzzerBeat_2`":

```
//A sketch to demonstrate the tone() function

//Specify digital pin on the Arduino that the positive lead of
piezo buzzer is attached.
const int piezoPin = 8;

void setup() {

}//close setup

void loop() {

  /*Tone needs 2 arguments, but can take three
    1) Pin#
    2) Frequency - this is in hertz (cycles per second) which
determines the pitch of the noise made
    3) Duration - how long the tone plays
  */
  tone(piezoPin, 450);
  delay(500);
  noTone(piezoPin);
  delay(500);
}
```

6. You can also try increasing the melody using a for-loop. Save it as “`TonePiezoBuzzerBeat_3`”:

```
//A sketch to demonstrate the tone() function

//Specify digital pin on the Arduino that the positive lead of
piezo buzzer is attached.
const int piezoPin = 8;

void setup() {

}//close setup

void loop() {

  /*Tone needs 2 arguments, but can take three
    1) Pin#
    2) Frequency - this is in hertz (cycles per second) which
determines the pitch of the noise made
    3) Duration - how long the tone plays
  */
  for (int i=31; i<10000; i++) {
    tone(piezoPin, i, 1000);
    delay(10);
  }
}
```

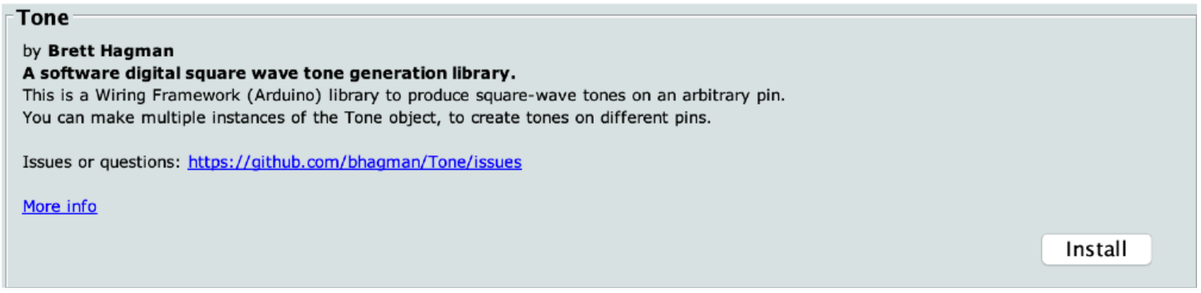
7. You can also explore the Super Mario Theme Song. One approach is to use the library Tone. Create a new file. Save it as “”Mario” and copy the following code:

```
#include <Tone.h>
Tone tone1;

void setup() {
  // put your setup code here, to run once:
  tone1.begin(8); // Playback on Pin 11, change to whatever you may
need
}

void loop() {
  // put your main code here, to run repeatedly:
  tone1.play(660,100);
  delay(75);tone1.play(660,100);
  delay(150);tone1.play(660,100);
  delay(150);tone1.play(510,100);
  delay(50);tone1.play(660,100);
```

```
        delay(150);tone1.play(770,100);
        delay(275);tone1.play(380,100);
        delay(287);tone1.play(510,100);
        delay(225);tone1.play(380,100);
        delay(200);tone1.play(320,100);
        delay(250);tone1.play(440,100);
        delay(150);tone1.play(480,80);
        delay(165);tone1.play(450,100);
        delay(75);tone1.play(430,100);
        delay(150);tone1.play(380,100);
        delay(100);tone1.play(660,80);
        delay(100);tone1.play(760,50);
        delay(75);tone1.play(860,100);
        delay(150);tone1.play(700,80);
        delay(75);tone1.play(760,50);
        delay(175);tone1.play(660,80);
        delay(150);tone1.play(520,80);
        delay(75);tone1.play(580,80);
        delay(75);tone1.play(480,80);
        delay(175);tone1.play(510,100);
        delay(275);tone1.play(380,100);
        delay(200);tone1.play(320,100);
        delay(250);tone1.play(440,100);
        delay(150);tone1.play(480,80);
        delay(165);tone1.play(450,100);
        delay(75);tone1.play(430,100);
        delay(150);tone1.play(380,100);
        delay(100);tone1.play(660,80);
        delay(100);tone1.play(760,50);
        delay(75);tone1.play(860,100);
        delay(150);tone1.play(700,80);
        delay(75);tone1.play(760,50);
        delay(175);tone1.play(660,80);
        delay(150);tone1.play(520,80);
        delay(75);tone1.play(580,80);
        delay(75);tone1.play(480,80);
        delay(250);tone1.play(500,100);
        delay(150);tone1.play(760,100);
        delay(50);tone1.play(720,100);
        delay(75);tone1.play(680,100);
        delay(75);tone1.play(620,150);
        delay(150);tone1.play(650,150);
        delay(150);tone1.play(380,100);
        delay(75);tone1.play(430,100);
        delay(75);tone1.play(500,100);
        delay(150);tone1.play(430,100);
        delay(75);tone1.play(500,100);
        delay(50);tone1.play(570,100);
        delay(110);tone1.play(500,100);
        delay(150);tone1.play(760,100);
        delay(50);tone1.play(720,100);
        delay(75);tone1.play(680,100);
        delay(75);tone1.play(620,150);
        delay(150);tone1.play(650,200);
        delay(150);tone1.play(1020,80);
        delay(150);tone1.play(1020,80);
        delay(75);tone1.play(1020,80);
        delay(150);tone1.play(380,100);
        delay(150);tone1.play(500,100);
        delay(150);tone1.play(760,100);
        delay(50);tone1.play(720,100);
        delay(75);tone1.play(680,100);
        delay(75);tone1.play(620,150);
        delay(150);tone1.play(650,150);
        delay(150);tone1.play(380,100);
        delay(75);tone1.play(430,100);
        delay(75);tone1.play(500,100);
        delay(150);tone1.play(430,100);
        delay(75);tone1.play(500,100);
        delay(50);tone1.play(570,100);
        delay(110);tone1.play(500,100);
        delay(150);tone1.play(760,100);
        delay(50);tone1.play(720,100);
        delay(75);tone1.play(680,100);
        delay(75);tone1.play(620,150);
        delay(150);tone1.play(650,200);
        delay(150);tone1.play(1020,80);
        delay(150);tone1.play(1020,80);
        delay(75);tone1.play(1020,80);
        delay(150);tone1.play(380,100);
        delay(150);tone1.play(500,100);
        delay(150);tone1.play(760,100);
        delay(50);tone1.play(720,100);
        delay(75);tone1.play(680,100);
        delay(75);tone1.play(620,150);
        delay(150);tone1.play(650,150);
        delay(150);tone1.play(380,100);
        delay(75);tone1.play(430,100);
        delay(75);tone1.play(500,100);
        delay(150);tone1.play(430,100);
        delay(75);tone1.play(500,100);
        delay(50);tone1.play(570,100);
        delay(210);tone1.play(585,100);
        delay(275);tone1.play(550,100);
        delay(210);tone1.play(500,100);
        delay(180);tone1.play(380,100);
        delay(150);tone1.play(500,100);
        delay(150);tone1.play(500,100);
        delay(75);tone1.play(500,100);
        delay(150);tone1.play(500,60);
        delay(75);tone1.play(500,80);
        delay(150);tone1.play(500,60);
        delay(175);tone1.play(500,80);
        delay(75);tone1.play(580,80);
        delay(175);tone1.play(660,80);
        delay(75);tone1.play(500,80);
        delay(150);tone1.play(430,80);
        delay(75);tone1.play(380,80);
        delay(300);tone1.play(500,60);
        delay(75);tone1.play(500,80);
        delay(150);tone1.play(500,60);
        delay(175);tone1.play(500,80);
```

```
    delay(75);tone1.play(580,80);
    delay(75);tone1.play(660,80);
    delay(225);tone1.play(870,80);
    delay(162);tone1.play(760,80);
    delay(300);tone1.play(500,60);
    delay(75);tone1.play(500,80);
    delay(150);tone1.play(500,60);
    delay(175);tone1.play(500,80);
    delay(75);tone1.play(580,80);
    delay(175);tone1.play(660,80);
    delay(75);tone1.play(500,80);
    delay(150);tone1.play(430,80);
    delay(75);tone1.play(380,80);
    delay(300);tone1.play(660,100);
    delay(75);tone1.play(660,100);
    delay(150);tone1.play(660,100);
    delay(150);tone1.play(510,100);
    delay(50);tone1.play(660,100);
    delay(150);tone1.play(770,100);
    delay(225);tone1.play(380,100);
    //tells the user it is over and delays it a little before going
  to the top again
    delay(1000);
    tone1.play(440,200);
    delay(200);
    delay(200);
    tone1.play(440,400);
    delay(200);
    delay(200);
    delay(5000);
  }
```

As you can see in the first line of the code `#include <Tone.h>`, you need to import the `Tone` library. For that, click on `Sketch > Import library`. Search for `Tone` and install it. You can find more info about installing Arduino libraries here: https://docs.arduino.cc/hacking/software/Libraries


Tone

by **Brett Hagman**
A software digital square wave tone generation library.
This is a Wiring Framework (Arduino) library to produce square-wave tones on an arbitrary pin.
You can make multiple instances of the Tone object, to create tones on different pins.

Issues or questions: https://github.com/bhagman/Tone/issues

More info

You can then verify the code and upload it to the board.

What is this library doing for you? The code is available on GitHub for your inspection (https://github.com/bhagman/Tone), as well as under the Arduino/libraries folder. But you should not worry too much at this stage, just focus on understanding how to use the library.

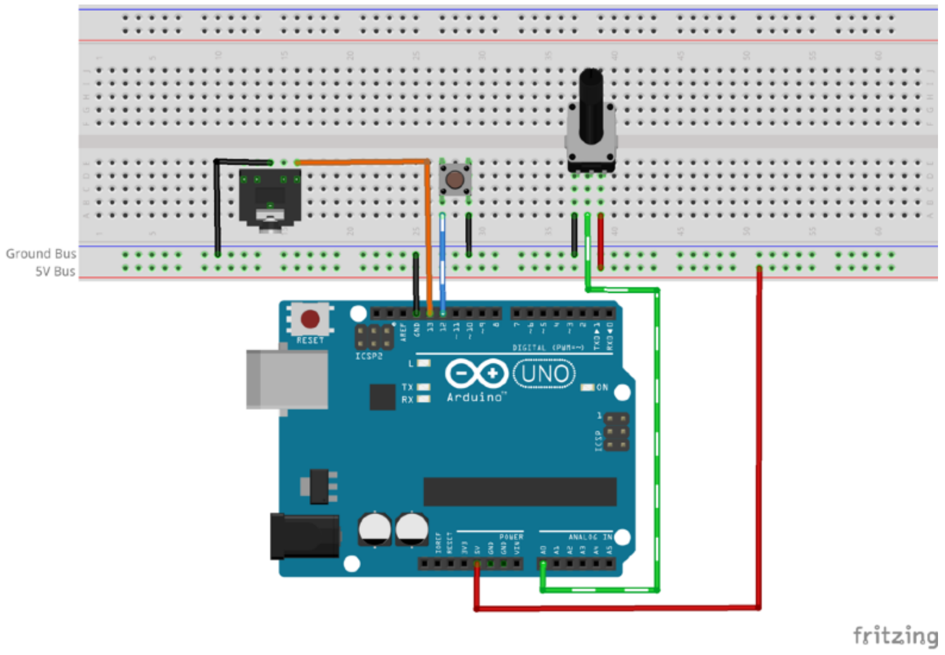
See the examples provided in `File > Examples > Tone`, especially the RTTTL examples.

8. Another way of by exploring the Super Mario Theme Song is downloading the code from the following link, save it as `“Mario_2”` and uploading it to the above circuit configuration: https://create.arduino.cc/projecthub/jrance/super-mario-theme-song-w-piezo-buzzer-and-arduino-1cc2e4

What is this code doing? What are the public constants defined at the beginning of the code used for? What is the function `buzz()` doing exactly?

## A DIGITAL OSCILLATOR

Another way of making sounds with Arduino is taking advantage of the possibility of creating a square wave using a digital pin and the two states of on/off or HIGH/LOW. We have already seen this with the code `“Mario_2”`. Let’s explore it further!



Time: 30’

1. Build the following circuit with the following parts:
    1. Solderless breadboard.
    2. Jumper wires.
    3. One tactic pushbutton switch.
    4. Jack connection.
    5. Potentiometer (10K or 100K).
    6. Battery-powered speaker / mini-amp / guitar amp.
    7. Jack wire.
2. The connections need to be connected as follows (unplug the USB cord from your Arduino Uno so it is not powered while you make your connections):
    1. Hook up an audio jack to pin 13, a button to pin 12, and a potentiometerter to pin A0.
    2. The signal/tip connection of the jack is connected to pin 13 and the ground is connected to a ground bus of the breadboard. Remember Nicolas Collins’ rule #10: “Every audio connection consists of two parts: the signal and a ground reference.”
    3. The button is connected to pin 12 on one side and the ground bus on the other.
    4. The potentiometer is connected to A0 from its centre tab, the ground bus on the left, and the +5v bus on the right.
    5. The ground bus on the breadboard is connected to the “GND” pin on the Uno.

6. The +5v bus on the breadboard is connected to “5v” on the Uno.

3. Write the following code on the Arduino IDE software. Save it as `“DigitalOSC”`.

```
//CODE DigitalOSC ******
const int ledPin = 13;      //variable to represent LED Pin
const int periodKnob = A0; //variable for knob pin (A0 = analog in
pin 0)
int delayTime;              //variable for the delay time

void setup() {
  pinMode(ledPin, OUTPUT); //configure pin 13 as a digital output
}

void loop() {
  //set delay time equal to the current value read on analog pin 0
  delayTime = analogRead(periodKnob);

  //map the analog read range from 0-1023 to 10000-1
  //delayTime = map(delayTime, 0, 1023, 10000, 1);

  digitalWrite(ledPin, HIGH);     //set pin 13 to 5 volts
  delayMicroseconds(delayTime);   //pause program
  digitalWrite(ledPin, LOW);      //set pin 13 to 0 volts
  delayMicroseconds(delayTime);   //pause program
}
//**********END OF CODE
```

If you code uploads correctly, you should hear a square wave that changes pitch as you turn the knob.

Let’s analyse it line by line. The first three lines of code after the initial comment each declare variables.

This sketch uses variables for two purposes: to indicate which pins external hardware is connected to and to store a number that that we will use to change the delay time while the program is running. The first two variables have constant values. The third variable declaration is still declared as an int but the variable `delayTime` is not set equal to anything. We do this because the value of `delayTime` will be set and changed by functions in the code as it is running.

This circuit uses 2 of the 14 digital pins available on Arduino Uno, although this first example sketch only utilises 1 of these pins for now. The second connection will be programmed to read a button as we build on this code in the next example. Recall that each digital pin can be configured as either an input or an output, and all speak the two-word language of 0 or 5 volts. Pin 13 is configured as an output in the `setup()` function.

The mode is set using the following line of instruction: `pinMode(ledPin, OUTPUT);`
The word OUTPUT written in capital letters, tells the IDE to set up the microcontroller pin for being set to a “HIGH” state of 5 volts or a “LOW” state of 0 volts (also known as ground). As we’ve seen in a previous lab worksheet, this can illuminate an LED (if you add a resistor in series–see Nicolas Collins’ Rule #22: “Always use a resistor when powering an LED, otherwise the circuit and/or LED might blow out”).

This same pin can be hooked to an audio jack and used as our sound source. If the pin is not set up as an output in the setup() function, the LED will look very dim or not light at all. If you run into trouble with a digital output, double-check that its mode has been properly defined in the `setup()` function.

We toggle the pin's value inside of the `loop()` function. When a pin is set up as an output, we use the function `digitalWrite()` to tell the microcontroller to either set the output to 5 volts (`digitalWrite(ledPin, HIGH)`) or to 0 (`digitalWrite(ledPin, LOW)`). A blinking light or rectangular audio waveform can be produced if we toggle the pin high, wait a short amount of time using the `delay()` function, then toggle the pin low, `delay()`, and repeat.

The Arduino Uno can only directly produce a rectangular wave from its output pins. There are no analogue output pins, so no voltage in between 0 and 5 volts can be directly generated by the Arduino Uno. Analog-like output effects can be achieved using Pulse Width Modulation (PWM), and true analogue outputs are possible using additional circuitry.

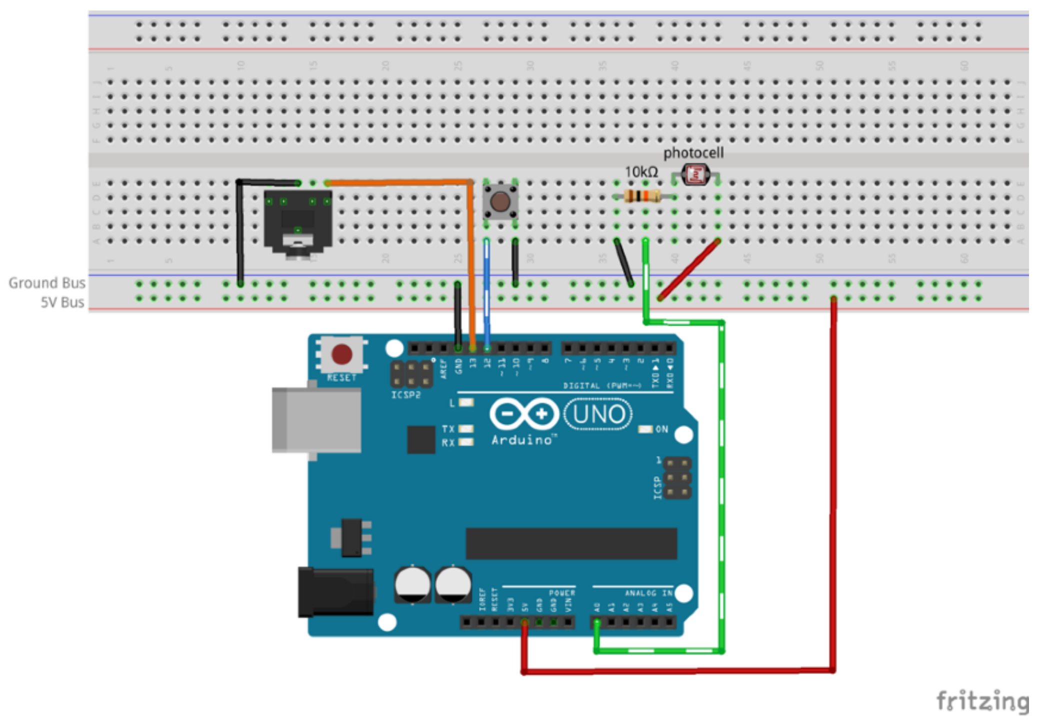
This example also features a potentiometer wired up to the Uno through the analogue input header. These six pins are dedicated analogue inputs. They cannot be configured as analogue outputs, so it is not necessary to set the pin mode as we did for the digital pins.

The Uno can read the voltage on its analogue input pins using the function `analogRead()`. An argument in the parenthesis tells the Uno which analog input pin to read (in our case `A0` represented by the variable `periodKnob`). The `analogRead()` function returns a number in the range of 0–1023 (this is the range if the 10-bit analogue-to-digital converter in the Uno).

In a single line of instruction, `delayTime = analogRead(periodKnob)`, we set our variable `delayTime` equal to the result of the function `analogRead(periodKnob)`.

The next line of instruction, `delayMicroseconds(delayTime)`, will set the length of the pause relative to the position of the potentiometer. Since the length of the delays is responsible for the frequency of the oscillator, this is how the pot is able to control the frequency.

The potentiometer is set up as a voltage divider. Other sensors could be used here as well. For example a photocell, pressure sensors, and so on, as alternative of voltage dividers.



Note: Many voltage divider sensors using fixed resistors will not be able to provide voltages that cover the entire range from 0 volts to 5 volts. It can be useful to see the actual values that are making it into your code. To do this, you must open a line of communication between the microcontroller and the computer. For an example of how to do this, start with the built-in example in `File > Examples > 01.Basics > AnalogReadSerial`. Once you know the actual range being sent, you can use the `map()` function to quickly scale to your desired target range.

Additionally, there are many novel, off-the-shelf sensor that have analogue outputs that can be substituted into any sketch that uses `analogRead()`: joysticks; distance, temperature, and sound level sensors; air pressure sensors for breath control; etc. The key is to find sensors that provide an analogue output voltage.

You will notice that turning the knob changes the pitch, but the range is fairly small and the direction is the opposite of what might seem most intuitive. You can change the operating direction by swapping the +5v and ground wire connections from the potentiometer. But we are going to leave the wiring unchanged and both reverse the direction and at the same time scale the range of numbers the knob provides to the code.

4. Comment out the following line:

```
//CODE DigitalOSC ******
const int ledPin = 13;      //variable to represent LED Pin
const int periodKnob = A0;  //variable for knob pin (A0 = analog in
pin 0)
int delayTime;              //variable for the delay time

void setup() {
  pinMode(ledPin, OUTPUT); //configure pin 13 as a digital output
}

void loop() {
  //set delay time equal to the current value read on analog pin 0
  delayTime = analogRead(periodKnob);

  //map the analog read range from 0-1023 to 10000-1
  delayTime = map(delayTime, 0, 1023, 10000, 1);

  digitalWrite(ledPin, HIGH);      //set pin 13 to 5 volts
  delayMicroseconds(delayTime);    //pause program
  digitalWrite(ledPin, LOW);       //set pin 13 to 0 volts
  delayMicroseconds(delayTime);    //pause program
}
//**********END OF CODE
```

5. Reupload the code. You will see that the knob now has increased range and is operating in the conventional direction. You can experiment with changing the las two values in the map function to experiment with different range and direction of settings.

## A BUTTON OSCILLATOR

Time: 15’

Let's use the button now. One side of the button is connected directly to ground, and the other side is attached to a digital pin. We will need to set the digital pin's mode in the `setup()` to allow the microcontroller to read the voltage on the pin. We will use the `INPUT_PULLUP` mode. The pin will always read `HIGH` unless the button is pressed, thereby connecting the pin to ground and causing the button to read `LOW`.

To do something useful with the button, we will need to set up a control structure to check and see if the button is being pressed. Control structures can effectively create alternative paths and mini loops inside the main loop of the programme.

If the button is pressed (the pin reads LOW), we want to play a tone. In our sketch, we will use the "if/else" control structure:

```
if (digitalRead(buttonPin) == LOW) {
// play my tone here
}
else {
//do not play the tone
}
```

The "if statement" in this code uses a conditional statement to decide whether or note to execute the instructions that follow in the block between its opening and closing curly braces. If the microcontroller reads the button pin and sees that it is equal to "LOW", it will execute the code between the braces. If the condition is not met, the microcontroller will skip the instructions in the if statement and simply move on to execute whatever code follows the closing brace.

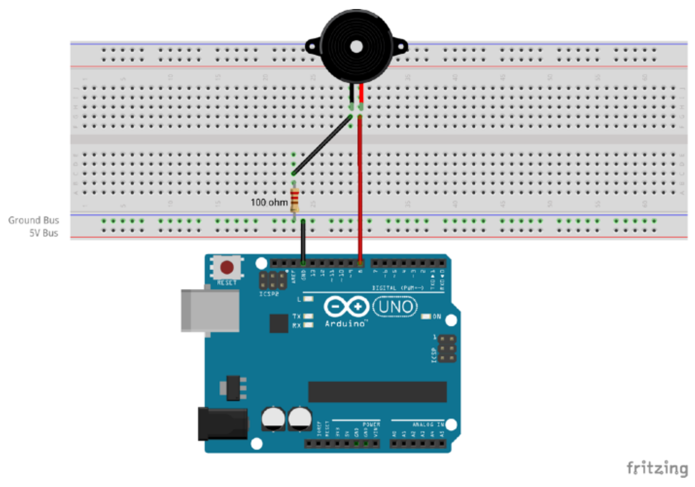
1. Without changing the circuit on the breadboard, create a new file. Write the following code on the Arduino IDE software. Save it as "ButtonOSC".

```
//CODE ButtonOSC ******

const int ledPin = 13;          //variable to represent LED Pin
const int buttonPin = 12;       //variable for the button pin
int periodKnob = A0;    //variable for knob pin (A = analog in)
int delayTime;          //variable for the delay time
//unsigned int means the integer doesn't go negative, so it can be
twice as big
unsigned int delayMax;  //variable for max delay time

void setup() {
  pinMode(ledPin, OUTPUT); //configure pin 13 as an output
  //INPUT_PULLUP sets the pin high. It gets pulled "low" by
  //connecting it to ground through a button
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
  //check the button. It will be low if pressed
  //If the button is pressed, execute the oscillator code:
  if (digitalRead(buttonPin) == LOW) {
    //set delay time equal to the current value read on analog pin
0
    delayTime = analogRead(periodKnob);
    //map the analog read range from 0 - 1023 to 10000 to 1
    delayTime = map(delayTime, 0, 1023, 10000, 1);

    digitalWrite(ledPin, HIGH);
    delayMicroseconds(delayTime); //shorter delay
    digitalWrite(ledPin, LOW);
    delayMicroseconds(delayTime);
  }

  else {
    //if the button is not pressed, execute the noise oscillator
    delayMax = analogRead(periodKnob);  //set delayMax to current
knob reading
    //map delayMax from 0 - 1023 range to 0 - 30000 range
    delayMax = map(delayMax, 0, 1023, 20, 30000);
    digitalWrite(ledPin, HIGH);   //toggle pin 13 to +5V
    //Pause the sketch for a random number of
    //microseconds between 0 and delayMax
    delayMicroseconds(random(delayMax));
    digitalWrite(ledPin, LOW);
    delayMicroseconds(random(delayMax));
  }

}
//**********END OF CODE
```

If the button is not pressed and the pin reads `HIGH`, the code does what is in the `else` block. This example code has a variation of the blink sketch in the `else` statement that produces a noisy random rectangular wave that is influenced by the knob whenever the button is not being pressed. Pressing the button will will cause the pin to read LOW and engage the oscillator code from before. In an if/else structure, only one of the statements gets activated each time through the loop. To disable the noise and leave only the button-operated oscillator and silence, comment out or simply delete the else statement and its curly braces and rue-load the code.

In place of buttons and switches, it is also possible to use the outputs of other circuits as the inputs to microcontroller and vice versa. Just make sure that no voltages go over 5 volts and the grounds of all circuits are connected.

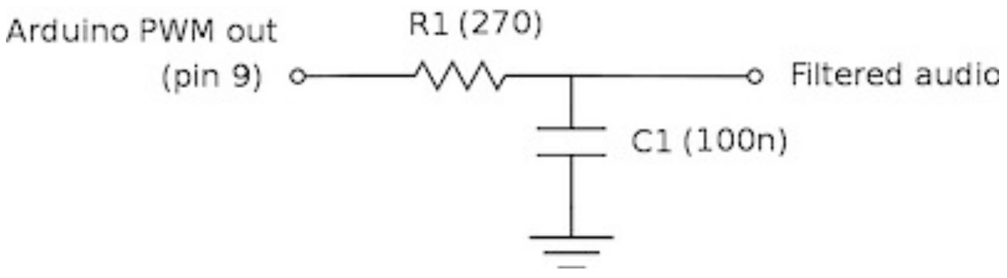
## MOZZI BASIC EXAMPLES



You can try the examples provided by the Mozzi library using the same circuit that we used in the Piezo buzzer exercises.

Alternatively, you can build a low pass audio filter to reduce aliasing at high frequencies as explained here: https://sensorium.github.io/Mozzi/learn/output/#low-pass-audio-filter

Low pass audio filter:



Time: 10'

1. Install the Mozzi library.
   1. Go to: https://github.com/sensorium/Mozzi
   2. Click on "Code" (top-left button) and select "Download ZIP".
   3. On the Arduino IDE, select `Sketch > Include library > Add .ZIP library…` and select the downloaded zip `"Mozzi-master.zip"`.
   4. Alternatively unzip "Mozzi-master.zip" and move the folder `"Mozzi-master"` to the folder `Arduino/libraries`.
2. Open the following examples from `File > Examples > Mozzi > 01. Basics >`
   1. `Control_Gain`
   2. `Vibrato`
3. Make sure the audio output is on pin 9.

For further documentation on the library go to: https://sensorium.github.io/Mozzi/doc/html/index.html

## BLOGGING

Time: 15'

1. **Complete your blog post.**