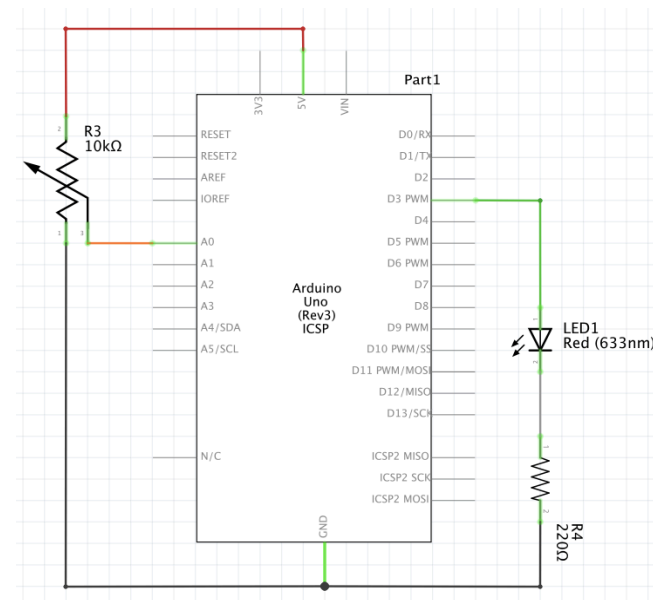
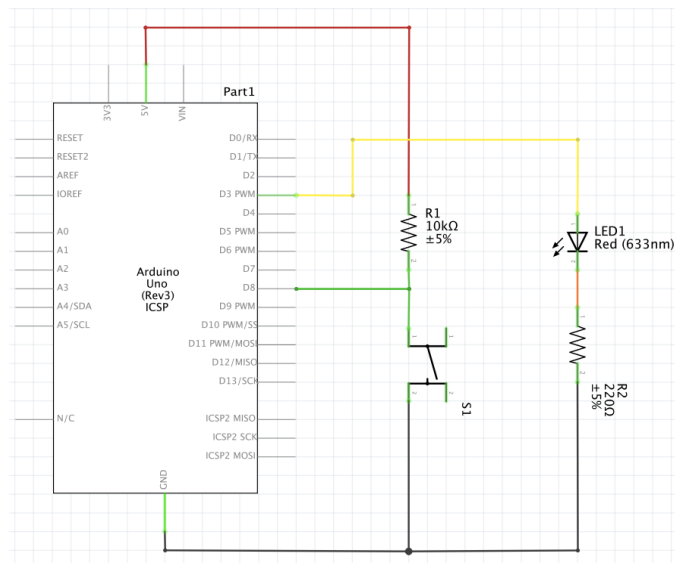


ELECTRONICS 1

ELECTRONICS FOR INTERACTIVE MEDIA DESIGN
LES 4

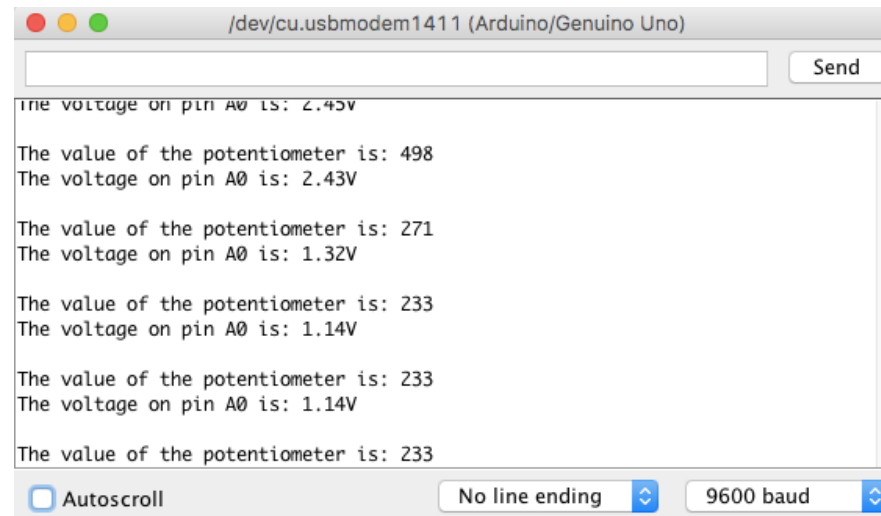
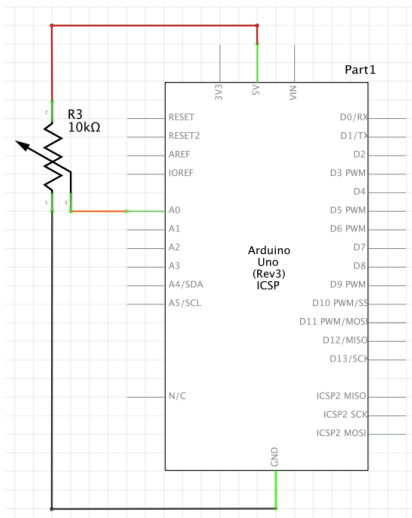
EMMA PARESCHI

FROM THE LAST LESSON



DATA TYPE (KIND OF VARIABLES)

WHEN WE READ THE ANALOG VALUE OF THE POTENTIOMETER,
WE CONVERTED THE READING IN REAL VOLTAGE.



IN THE CODE WE DEFINED TWO KIND OF VARIABLES: INT AND FLOAT

```
int pot_value = 0;
float pot_voltage = 0;
```

DATA TYPE (KIND OF VARIABLE)

Numeric type	Bytes	Range	Use
int	2	-32768 to 32767	Represents positive and negative integer values.
unsigned int	2	0 to 65535	Represents only positive values; otherwise, similar to int.
long	4	-2147483648 to 2147483647	Represents a very large range of positive and negative values.
unsigned long	4	4294967295	Represents a very large range of positive values.
float	4	3.4028235E+38 to -3.4028235E+38	Represents numbers with fractions; use to approximate real- world measurements
double	4	Same as float	In Arduino, double is just another name for float.
boolean	1	false (0) or true (1)	Represents true and false values.
char	1	-128 to 127	Represents a single character. Can also represent a signed value between -128 and 127.
byte	1	0 to 255	Similar to char, but for unsigned values.

VARIABLE SCOPE

GLOBAL AND LOCAL VARIABLE

Variables in the C programming language, which Arduino uses, have a property called scope.

Scopes: GLOBAL and LOCAL

Global variable is one that can be seen by every function in a program.

Local variables are only visible to the function in which they are declared.

```
int led_pin = 3; // any function will see this variable
int sensor_value; // any function will see this variable

void setup(){
  // ...
}

void loop(){
  int i; // "i" is only "visible" inside of "loop"
  float f; // "f" is only "visible" inside of "loop"
  // ...
}
```

INCREMENT

You want to increase or decrease the value of a variable
+=
-=

```
int myValue = 0;

myValue = myValue + 1; // this adds one to the variable myValue
myValue += 1;          // this does the same as the above

myValue = myValue - 1; // this subtracts one from the variable myValue
myValue -= 1;          // this does the same as the above

myValue = myValue + 5; // this adds five to the variable myValue
myValue += 5;          // this does the same as the above
```

INCREMENT

You want to increase or decrease the value of a variable:

++

--

x++; // increment x by one and returns the old value of x

++x; // increment x by one and returns the new value of x

x-- ; // decrement x by one and returns the old value of x

--x ; // decrement x by one and returns the new value of x

```
int x = 0;
```

```
int y = 0;
```

```
y = x++; // x = 0; y = 1
```

```
y = ++x; // x = 1; y = 1
```

FOR

The FOR statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The FOR statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

```
for (initialisation; condition; increment) {  
  //statement(s);  
}
```

The diagram illustrates the components of a for loop using the example code: `for(int x = 0; x < 100; x++){ println(x); // prints 0 to 99 }`. Annotations include: 'parenthesis' with a yellow arrow pointing to the opening brace; 'declare variable (optional)' with a blue arrow pointing to 'int x'; 'initialize' with a purple arrow pointing to '= 0'; 'test' with a purple arrow pointing to '< 100'; 'increment or decrement' with a purple arrow pointing to 'x++'; and a yellow curved arrow indicating the loop's repetition from the end back to the start.

```
for(int x = 0; x < 100; x++){  
    println(x); // prints 0 to 99  
}
```


IMPLEMENTATION OF INCREMENTAL: FOR

To understand how FOR statement works, you can simply it as much as possible and print on Serial Monitor the value of the counter

```
void setup(){  
  Serial.print(9600);  
}  
  
void loop()  
{  
  for (int i=0; i <= 255; i++){  
    Serial.print(i);  
    delay(10);  
  }  
}
```

IMPLEMENTATION OF INCREMENTAL: FOR

You want to increment the brightness of a LED.

```
int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10

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

void loop()
{
  for (int i=0; i <= 255; i++){
    analogWrite(PWMpin, i);
    delay(10);
  }
}
```

ARRAY

An array is a collection of variables that are accessed with an index number.

To create (or declare) an array

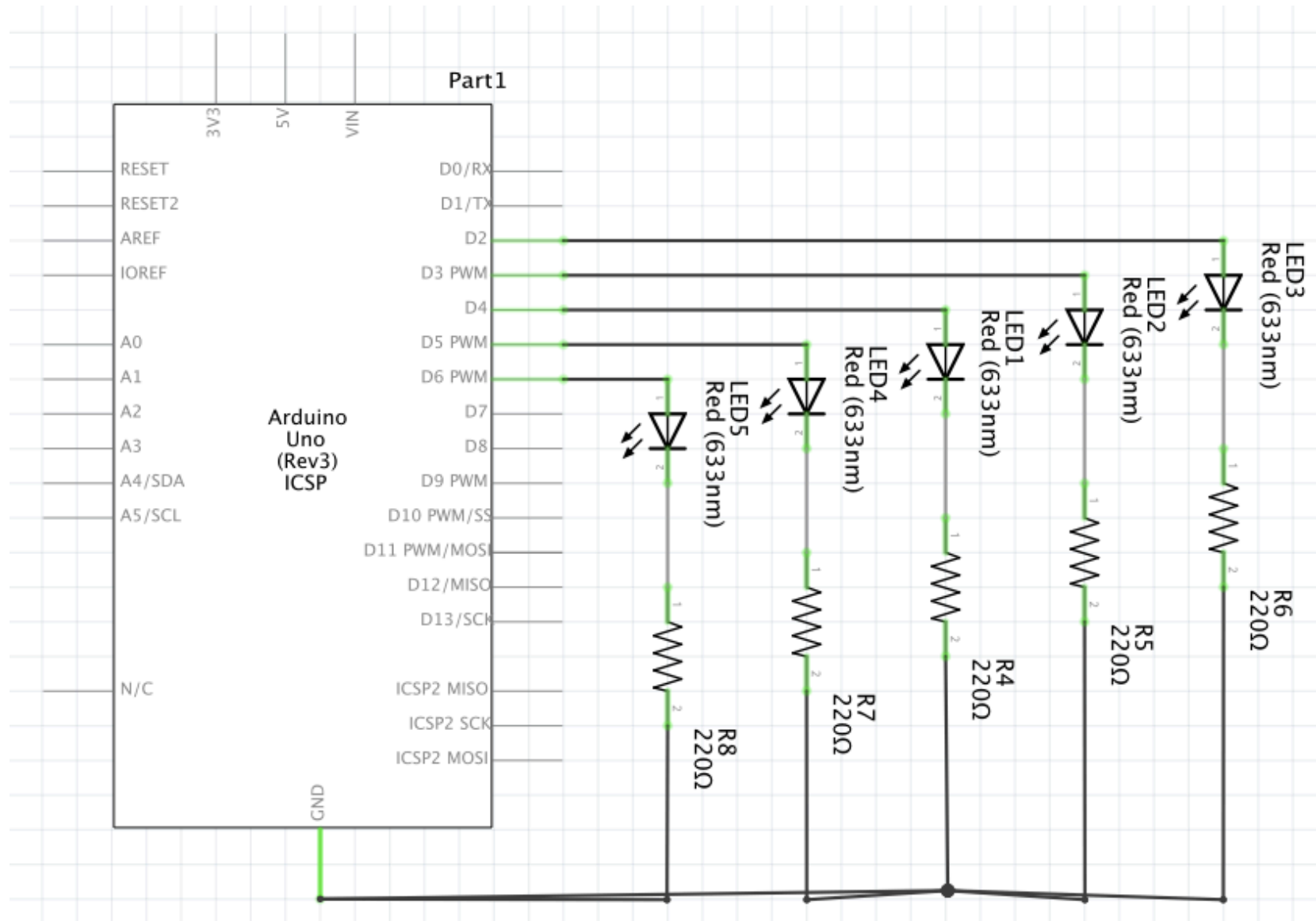
```
int myInts[6];  
int myPins[] = {2, 4, 8, 3, 6};  
int mySensVals[6] = {2, 4, -8, 3, 2};
```

Accessing an Array

Arrays are zeroindexed, that is, referring to the array initialization above, the first element of the array is at index 0,

```
int myArray[10]={9,3,2,4,3,2,7,8,9,11};  
  // myArray[9]   contains 11  
  // myArray[10]  is invalid and contains random information (other memory address)
```

LED ARRAY (AND FOR LOOP)



LED ARRAY - SKETCH 1

led_no_array

```
const int Led_pin_1 = 2;
const int Led_pin_2 = 3;
const int Led_pin_3 = 4;
const int Led_pin_4 = 5;
const int Led_pin_5 = 6;

void setup() {
  pinMode(Led_pin_1, OUTPUT);
  pinMode(Led_pin_2, OUTPUT);
  pinMode(Led_pin_3, OUTPUT);
  pinMode(Led_pin_4, OUTPUT);
  pinMode(Led_pin_5, OUTPUT);
}

void loop() {
  digitalWrite(Led_pin_1, HIGH);
  delay(100);
  digitalWrite(Led_pin_1, LOW);
  digitalWrite(Led_pin_2, HIGH);
  delay(100);
  digitalWrite(Led_pin_2, LOW);
  digitalWrite(Led_pin_3, HIGH);
  delay(100);
  digitalWrite(Led_pin_3, LOW);
  digitalWrite(Led_pin_4, HIGH);
  delay(100);
  digitalWrite(Led_pin_4, LOW);
  digitalWrite(Led_pin_5, HIGH);
  delay(100);
  digitalWrite(Led_pin_5, LOW);
}
```

LED ARRAY - SKETCH 2

```
led_array
/*
 * Emma Pareschi
 * October 2017
 * LED bar graph
 */

// these constants won't change:
const int ledCount = 5;    // the number of LEDs in the bar graph

int ledPins[] = {
  2, 3, 4, 5, 6
};    // an array of pin numbers to which LEDs are attached

void setup() {
  // loop over the pin array and set them all to output:
  for (int i = 0; i < ledCount; i++) {
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop() {
  // loop over the LED array:
  for (int j = 0; j < ledCount; j++) {
    // if the array element's index is less than ledLevel,
    // turn the pin for this element on:
    digitalWrite(ledPins[j], HIGH);
    delay(100);
    digitalWrite(ledPins[j], LOW);
  }
}
```

GENERATING SOUNDS: TRANSDUCER

It converts electrical energy into sound.

The term transducer here describes a noise-creating device that is driven by external electronics.

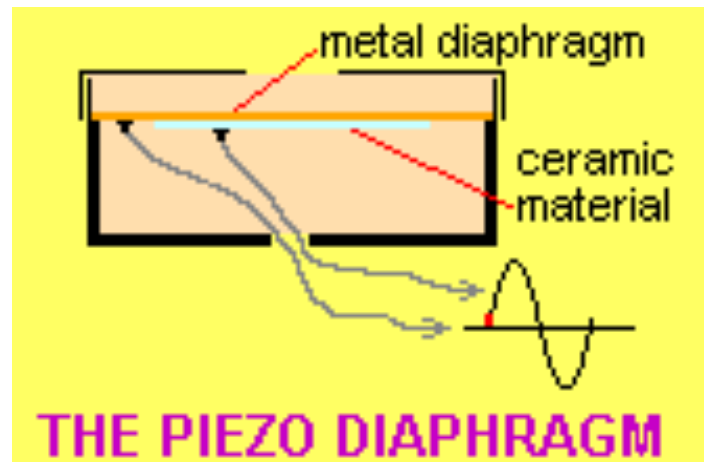
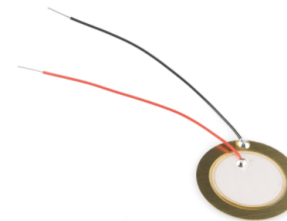
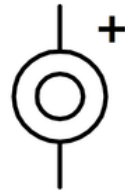
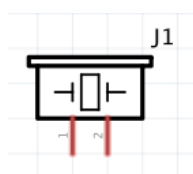
An audio transducer is a device that can create an **alert**. It requires an AC signal that is supplied by external electronics, and in its simplest form may be referred to as a **buzzer** or a **beeper**.

Audio alerts are used in microwave ovens, washer/dryers, automobiles, gasoline pumps, security devices, toys, phones, and many other consumer products. They are often used in conjunction with touch pads, to provide audio confirmation that a tactile switch has been pressed.

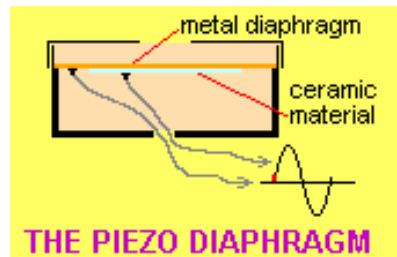
Variants: Electromagnetic, Piezoelectric , Ultrasonic

TRANSDUCER - PIEZOELECTRIC

A piezoelectric transducer contains a diaphragm consisting of a thin brass disc on which is mounted a **ceramic wafer**. When an AC signal is applied between the piezoelectric wafer and the disc, the disc flexes at that frequency.



FREQUENCY



Frequency is the number of occurrences of a repeating event per **unit of time**.

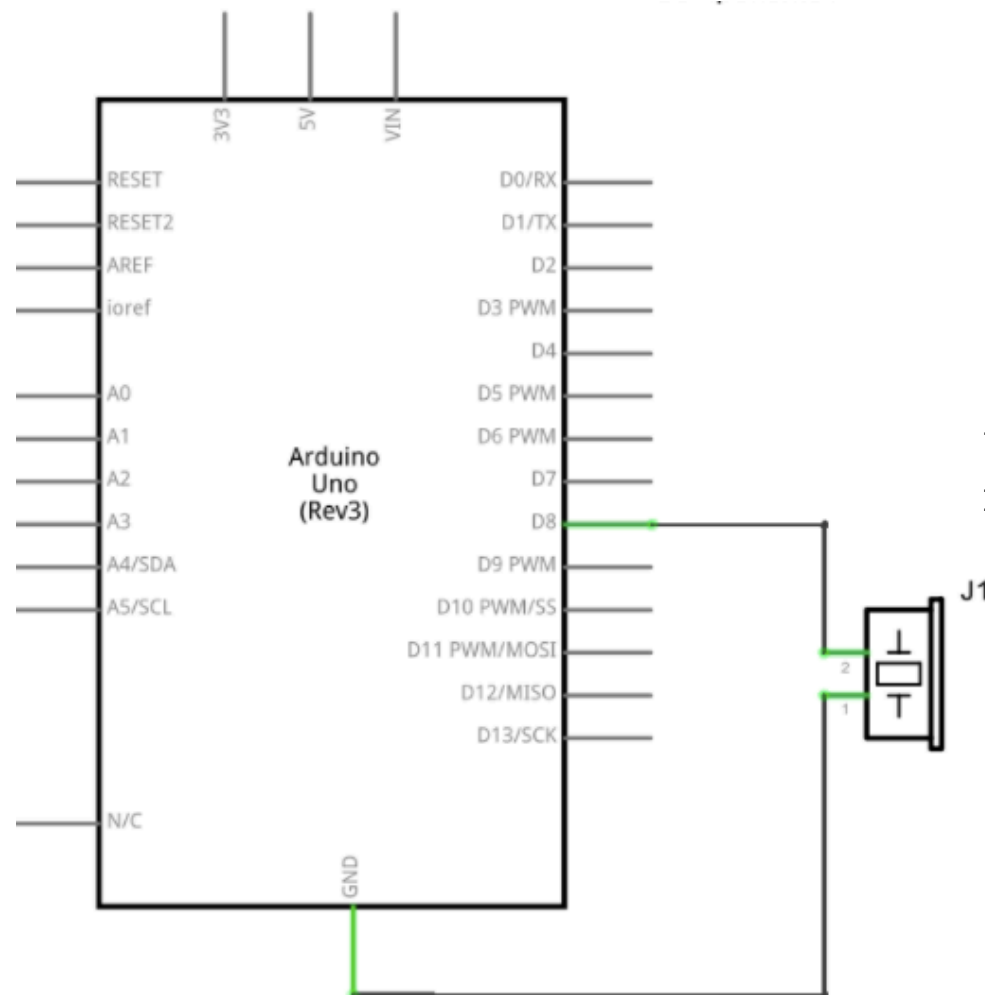
Audio frequency is measured in Hertz, abbreviated Hz.

The H in Hz is capitalised because it refers to a real name. One thousand Hertz can be written as 1 kiloHertz, almost always abbreviated as 1kHz (note that the k is lowercase).

The human ear is often described as being able to detect sounds between 20Hz and 20kHz, although the ability to hear sounds above 15kHz is relatively unusual and diminishes naturally with age. Sensitivity to all frequencies can be impaired by long-term exposure to loud noise.

The most common frequencies applied to audio transducers range between 3kHz and 3.5kHz. Piezoelectric elements are inefficient for generating sounds below 1kHz.

PIEZO SCHEMATIC



THE PIEZO IS CONNECTED
BETWEEN PIN 8 AND GROUND

PIEZO SKETCH

piezo §

```
/*Emma pareschi  
 * October 2017  
 * I generate a sound with a piezo element  
 * and I apply a frequency of 1000Hz  
 */
```

```
const int buzzer = 8; //buzzer to arduino pin 9
```

```
void setup(){
```

```
  pinMode(buzzer, OUTPUT); // Set buzzer - pin 9 as an output
```

```
}
```

```
void loop(){
```

```
  tone(buzzer, 1000); // Send 1KHz sound signal
```

```
}
```

Function:
- Tone();

FUNCTION TONE

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

1. The pin number that you will use on the Arduino.
2. The frequency specified in hertz

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

1. The pin number that you will use on the Arduino.
2. The frequency specified in hertz
3. The duration of the tone in milliseconds (optional) - unsigned long

```
const int buzzer = 8;
```

```
tone(buzzer, 1000);
```

```
tone(buzzer, 1000, 5000);
```

NOTES:

- MIN FREQUENCY: 31Hz
- MAX FREQUENCY: 65535

FUNCTION TONE // noTONE

```
const int buzzer = 8;
```

```
tone(buzzer, 1000, 1000);  
delay(1000);
```


I WANT TO GENERATE DISTINCT BEATS LONG 1SEC
EVERY 1SEC.

I TRY TO USE DELAY.
BUT IT DOESN'T WORK.
WHY?

```
tone(buzzer, 3000, 500);  
delay(1000);
```

```
tone(peizoPin, 3000, 500)
```

```
delay(1000)
```



```
tone(buzzer, 1000); // Send 1KHz sound signal..  
delay(1000);        // ...for 1 sec  
noTone(buzzer);     // Stop sound..  
delay(1000);        // ...for 1sec
```

```
noTone( pin number);
```

Stops the generation
of a square wave
triggered by `tone()`

PIEZO SKETCH

piezo

```
/*Emma pareschi
 * October 2017
 * I generate a sound with a piezo element
 * and I apply a frequency of 1000Hz
 */

const int buzzer = 8; //buzzer to arduino pin 9

void setup(){

  pinMode(buzzer, OUTPUT); // Set buzzer - pin 9 as an output
}

void loop(){

  tone(buzzer, 1000); // Send 1KHz sound signal...
  delay(1000);        // ...for 1 sec
  noTone(buzzer);     // Stop sound...
  delay(1000);        // ...for 1sec
}
```

LIMITS OF THE TONE FUNCTION

1. You can't use `tone()` while **also using `analogWrite()`** on pins 3 or 11. If you do - you get some whacky results - neither will work like you expect. That's because the `tone()` function uses the same built in timer that `analogWrite()` does for pins 3 and 11. It's worth trying just hear the weird noises.
2. You cannot generate a tone lower than 31 HZ. You can pass values 31 and less to the `tone()` function, but it doesn't mean you will get a good representation of it.
3. The `tone()` function cannot be used by two separate pins at the same time. Let's say you have two separate piezo speakers, each on a different pin. You can't have them both play at the same time. One has to be on, and then the other. Furthermore, before you can have the other pin use the `tone()` function, you must call the `noTone()` function and "turn off" the tone from the previous pin.

MELODY

TO CREATE A MELODY YOU NEED NOTES. EACH NOTE IS DEFINED BY A FREQUENCY.

NOTE FREQUENCY

	C	C#	D	Eb	E	F	F#	G	G#	A	Bb	B
0	16.35	17.32	18.35	19.45	20.60	21.83	23.12	24.50	25.96	27.50	29.14	30.87
1	32.70	34.65	36.71	38.89	41.20	43.65	46.25	49.00	51.91	55.00	58.27	61.74
2	65.41	69.30	73.42	77.78	82.41	87.31	92.50	98.00	103.8	110.0	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185.0	196.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902

```
#define NOTE_C4 262  
#define NOTE_G3 196  
#define NOTE_A3 220  
#define NOTE_B3 247
```


MELODY

piezo_simple_melody

```
#define NOTE_C4 262
#define NOTE_G3 196
#define NOTE_A3 220
#define NOTE_B3 247

const int buzzer = 8;

// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};

void setup() {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // to calculate the note duration, take one second divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(buzzer, melody[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(buzzer);
  }
}

void loop() {
  // no need to repeat the melody.
}
```

MELODY

```
#include "pitches.h"

const int buzzer = 8;

// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
  4, 8, 8, 4, 4, 4, 4, 4
};

void setup() {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // to calculate the note duration, take one second divided by the no
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(buzzer, melody[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(buzzer);
  }
}

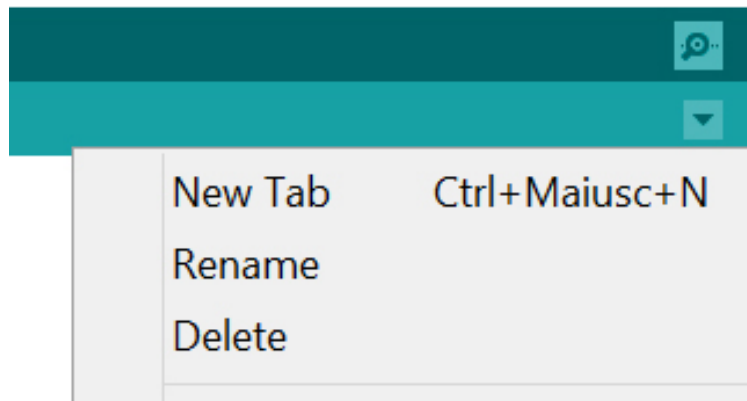
void loop() {
  // no need to repeat the melody.
}
```

```
piezo_melody pitches.h
/*****
 * Public Constants
 *****/

#define NOTE_B0 31
#define NOTE_C1 33
#define NOTE_CS1 35
#define NOTE_D1 37
#define NOTE_DS1 39
#define NOTE_E1 41
#define NOTE_F1 44
#define NOTE_FS1 46
#define NOTE_G1 49
#define NOTE_GS1 52
#define NOTE_A1 55
#define NOTE_AS1 58
#define NOTE_B1 62
#define NOTE_C2 65
#define NOTE_CS2 69
#define NOTE_D2 73
#define NOTE_DS2 78
#define NOTE_E2 82
#define NOTE_F2 87
#define NOTE_FS2 93
#define NOTE_G2 98
#define NOTE_GS2 104
#define NOTE_A2 110
#define NOTE_AS2 117
#define NOTE_B2 123
#define NOTE_C3 131
#define NOTE_CS3 139
#define NOTE_D3 147
#define NOTE_DS3 156
#define NOTE_E3 165
#define NOTE_F3 175
```

MELODY

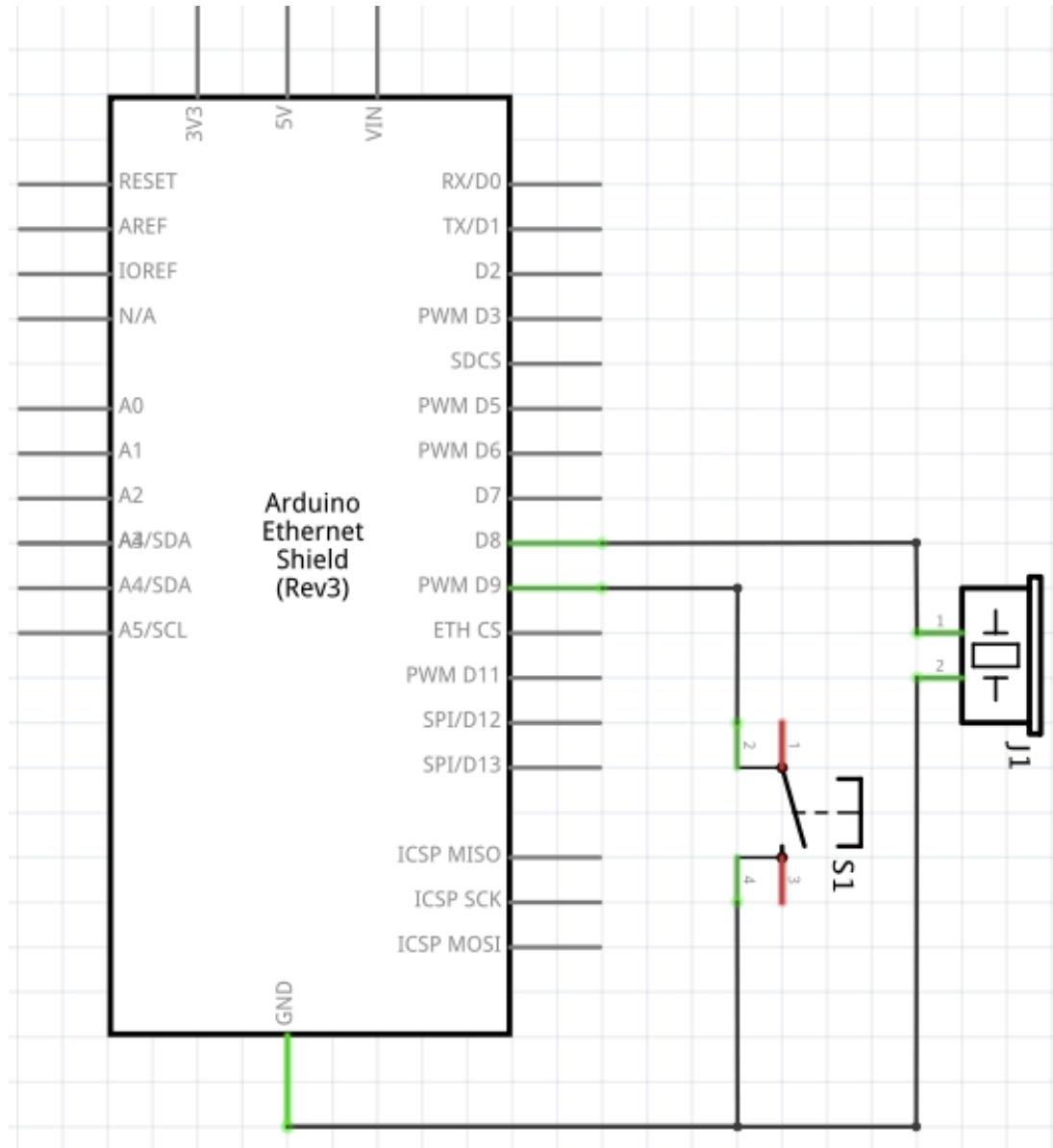
TO CREATE THE FILE PITCHES,
SELECT NEW TAB:



```
piezo_melody pitches.h
/*****
 * Public Constants
 *****/

#define NOTE_B0 31
#define NOTE_C1 33
#define NOTE_CS1 35
#define NOTE_D1 37
#define NOTE_DS1 39
#define NOTE_E1 41
#define NOTE_F1 44
#define NOTE_FS1 46
#define NOTE_G1 49
#define NOTE_GS1 52
#define NOTE_A1 55
#define NOTE_AS1 58
#define NOTE_B1 62
#define NOTE_C2 65
#define NOTE_CS2 69
#define NOTE_D2 73
#define NOTE_DS2 78
#define NOTE_E2 82
#define NOTE_F2 87
#define NOTE_FS2 93
#define NOTE_G2 98
#define NOTE_GS2 104
#define NOTE_A2 110
#define NOTE_AS2 117
#define NOTE_B2 123
#define NOTE_C3 131
#define NOTE_CS3 139
#define NOTE_D3 147
#define NOTE_DS3 156
#define NOTE_E3 165
#define NOTE_F3 175
```

PIEZO AND SWITCH - SCHEMATIC



THE PIEZO IS CONNECTED BETWEEN PIN 8 AND GROUND

THE TACT SWITCH IS CONNECTED BETWEEN PIN 9 AND GROUND AND IT USES THE INTERNAL PULL-UP

PIEZO AND SWITCH - SCHEMATIC

```
void setup() {  
  
    pinMode(sw_pin, INPUT_PULLUP);  
    pinMode(buzzer, OUTPUT);  
  
}  
  
void loop() {  
    sw_value = digitalRead(sw_pin);  
  
    if (sw_value == 0){  
        for (int thisNote = 0; thisNote < 8; thisNote++) {  
  
            // to calculate the note duration, take one second divided by the note type.  
            //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.  
            int noteDuration = 1000 / noteDurations[thisNote];  
            tone(buzzer, melody[thisNote], noteDuration);  
  
            // to distinguish the notes, set a minimum time between them.  
            // the note's duration + 30% seems to work well:  
            int pauseBetweenNotes = noteDuration * 1.30;  
            delay(pauseBetweenNotes);  
            // stop the tone playing:  
            noTone(buzzer);  
        }  
    }  
}
```

PIEZO AND SWITCH - SCHEMATIC

```
void setup() {  
  
    pinMode(sw_pin, INPUT_PULLUP);  
    pinMode(buzzer, OUTPUT);  
  
}  
  
void loop() {  
    sw_value = digitalRead(sw_pin);  
  
    if (sw_value == 0){  
        for (int thisNote = 0; thisNote < 8; thisNote++) {  
  
            // to calculate the note duration, take one second divided by the note type.  
            //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.  
            int noteDuration = 1000 / noteDurations[thisNote];  
            tone(buzzer, melody[thisNote], noteDuration);  
  
            // to distinguish the notes, set a minimum time between them.  
            // the note's duration + 30% seems to work well:  
            int pauseBetweenNotes = noteDuration * 1.30;  
            delay(pauseBetweenNotes);  
            // stop the tone playing:  
            noTone(buzzer);  
        }  
    }  
}
```

PIEZO AND SWITCH - SKETCH

```
piezo_sw_2 pitches.h
void setup() {
    pinMode(sw_pin, INPUT_PULLUP);
    pinMode(buzzer, OUTPUT);
}

void loop() {
    sw_value = digitalRead(sw_pin);

    if (sw_value == 0){
        play(); //call the function play
    }
}

//definition of the function play
void play(){
    for (int thisNote = 0; thisNote < 8; thisNote++) {

        // to calculate the note duration, take one second divided by the note type.
        //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
        int noteDuration = 1000 / noteDurations[thisNote];
        tone(buzzer, melody[thisNote], noteDuration);

        // to distinguish the notes, set a minimum time between them.
        // the note's duration + 30% seems to work well:
        int pauseBetweenNotes = noteDuration * 1.30;
        delay(pauseBetweenNotes);
        // stop the tone playing:
        noTone(buzzer);
    }
}
```

FUNCTION

Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a program.

Advantages:

- Help to stay organized. Often this helps to conceptualise the program.
- Functions codify one action in one place so that the function only has to be thought out and debugged once.
- Reduce chances for errors in modification, if the code needs to be changed.
- Functions make the whole sketch smaller and more compact because sections of code are reused many times.

They make it easier to reuse code in other programs by making it more modular, and as a nice side effect, using functions also often makes the code more readable.

FUNCTION

There are two required functions in an Arduino sketch, `setup()` and `loop()`. Other functions must be created outside the brackets of those two functions.

As an example, we will create a simple function to play a melody on a buzzer connected to pin 8: `play()`

```
void play(){
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // to calculate the note duration, take one second divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(buzzer, melody[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(buzzer);
  }
}
```

PIEZO AND SWITCH - SKETCH WITH FUNCTION

```
piezo_sw_2 pitches.h
void setup() {
    pinMode(sw_pin, INPUT_PULLUP);
    pinMode(buzzer, OUTPUT);
}

void loop() {
    sw_value = digitalRead(sw_pin);

    if (sw_value == 0){
        play(); //call the function play
    }
}

//definition of the function play
void play(){
    for (int thisNote = 0; thisNote < 8; thisNote++) {

        // to calculate the note duration, take one second divided by the note type.
        //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
        int noteDuration = 1000 / noteDurations[thisNote];
        tone(buzzer, melody[thisNote], noteDuration);

        // to distinguish the notes, set a minimum time between them.
        // the note's duration + 30% seems to work well:
        int pauseBetweenNotes = noteDuration * 1.30;
        delay(pauseBetweenNotes);
        // stop the tone playing:
        noTone(buzzer);
    }
}
```

FUNCTION

Anatomy of a C function

Datatype of data returned,
any C datatype.

"void" if nothing is returned.

Function name

Parameters passed to
function, any C datatype.

```
int myMultiplyFunction(int x, int y){  
    int result;  
    result = x * y;  
    return result;  
}
```

Return statement,
datatype matches
declaration.

Curly braces required.

FUNCTION - EXAMPLE

```
void setup(){
  Serial.begin(9600);
}

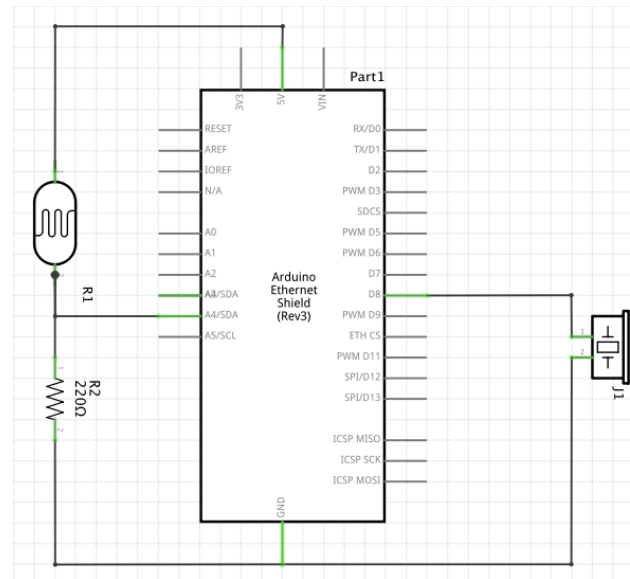
void loop() {
  int i = 2;
  int j = 3;
  int k;

  k = myMultiplyFunction(i, j); // k now contains 6
  Serial.println(k);
  delay(500);
}

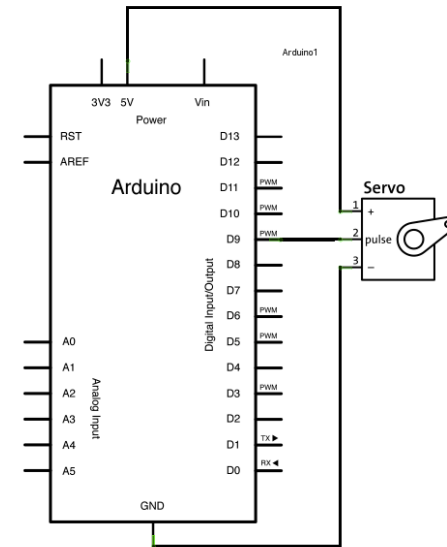
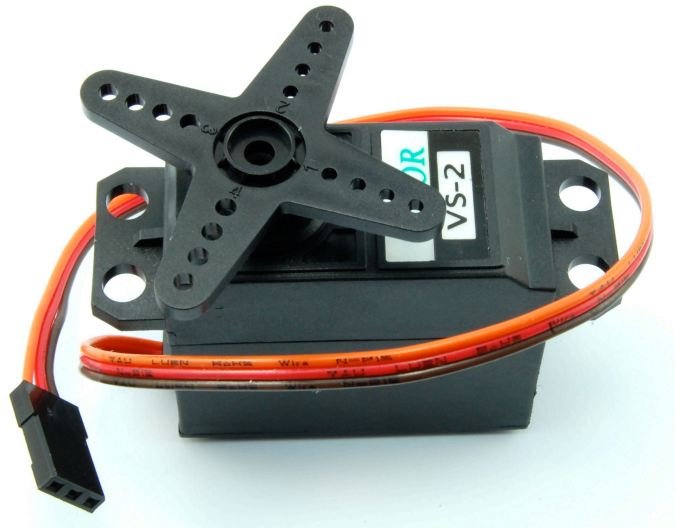
int myMultiplyFunction(int x, int y){
  int result;
  result = x * y;
  return result;
}
```

ASSIGNMENT

- READ HOW FUNCTIONS WORK: [HTTPS://WWW.ARDUINO.CC/EN/REFERENCE/FUNCTIONDECLARATION](https://www.arduino.cc/en/Reference/FunctionDeclaration)
- WRITE A CODE FOR A CIRCUIT ON WHICH YOU CONNECT AT LEAST ONE BUZZER AND ONE SENSOR.



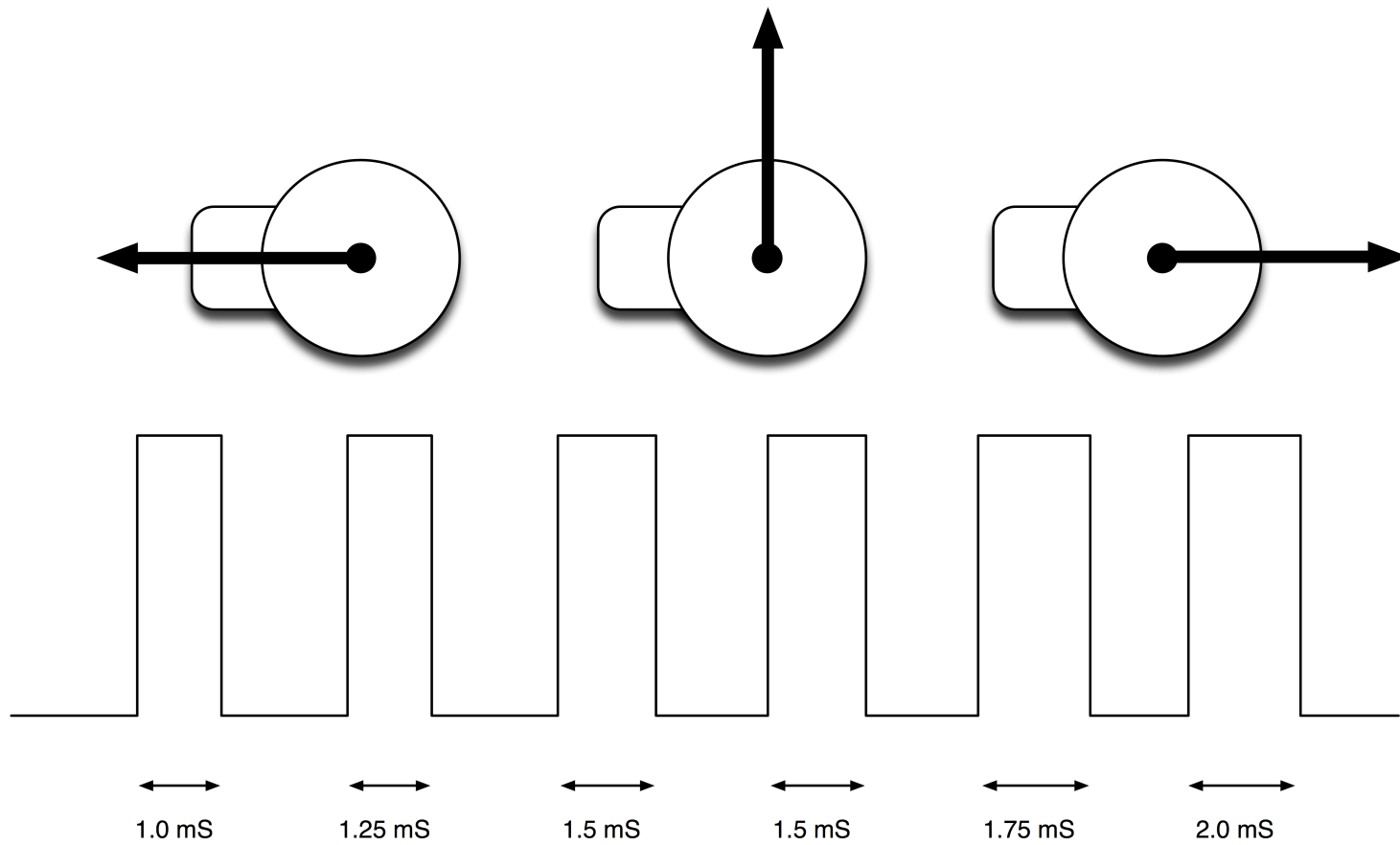
SERVO MOTOR



Three wires: power, ground, and signal.
The power wire is typically red.
The ground wire is typically black or brown
The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board.

Note that servos draw considerable power, so if you need to drive more than one or two, you'll probably need to power them from a separate supply (i.e. not the +5V pin on your Arduino). Be sure to connect the grounds of the Arduino and external power supply together.

SERVO MOTOR



SERVO MOTOR - EXAMPLE 1

```
#include <Servo.h>
```

```
Servo myservo; // create servo object to control a servo  
// twelve servo objects can be created on most boards
```

```
void setup() {  
  myservo.attach(9); // attaches the servo on pin 9 to the servo object  
}
```

```
void loop() {  
  myservo.write(0);           // tell servo to go to position 0  
  delay(10);                  // waits 15ms for the servo to reach the position  
  myservo.write(180);         // tell servo to go to position 180  
  delay(10);                  // waits 15ms for the servo to reach the position  
}
```

```
#include <Servo.h>
```

```
Servo "name";
```

```
"name".attach(pin);
```

```
"name".write(position);
```


SERVO MOTOR - EXAMPLE 2 - SWEEP

```
#include <Servo.h>

Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

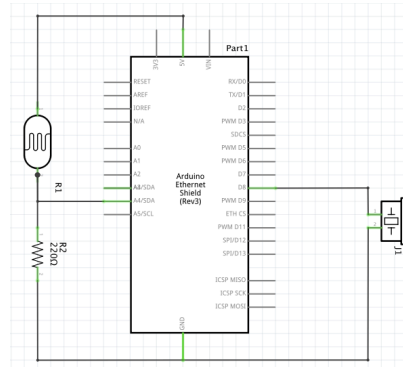
int pos = 0;    // variable to store the servo position

void setup() {
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```

ASSIGNMENT

- READ HOW FUNCTIONS WORK: [HTTPS://WWW.ARDUINO.CC/EN/REFERENCE/FUNCTIONDECLARATION](https://www.arduino.cc/en/Reference/FunctionDeclaration)
- WRITE A CODE FOR A CIRCUIT ON WHICH YOU CONNECT AT LEAST ONE BUZZER AND ONE SENSOR.



- MAKE A CIRCUIT USING A TACT SWITCH AND A SERVO MOTOR. WRITE A CODE TO CONTROL THE SERVO POSITION USING A TACT-SWITCH.

SOURCES AND LICENCE

Buzzer (pg. 249)

Encyclopedia of Electronic Components Volume 2, 1st Edition,
Charles Platt.

Arduino commands:

<https://www.arduino.cc/en/Reference/HomePage>

LICENCE

EXCEPT WHERE OTHERWISE NOTED, THIS WORK IS LICENSED UNDER:

[HTTPS://CREATIVECOMMONS.ORG/LICENSES/BY/4.0/](https://creativecommons.org/licenses/by/4.0/)

