Week 4 Lab Activities:
Microcontrollers: Arduino Inputs

Preparation:

1. Reading:
    a. Review in Wearable Electronics, Hartman, Ch.3: switches, Ch. 6: Digital Input, Analog Input, Ch. 7 (sensors)
    b. Review: "Pull Up Resistors" https://learn.sparkfun.com/tutorials/pull-up-resistors

Exercises:

What are we going to use in this workshop?

- **Solderless breadboard**
  !!! When you start to put components on your breadboard, avoid adding, removing, or changing components on a breadboard whenever the board is powered. You risk shocking yourself and damaging your components.
- **Arduino board**
- **USB cable**
- **LEDs**
- **Wires**
- **Resistors**
- **10 uF Capacitors**
- **Light Dependent Resistor (LDR)**
- **Piezo Buzzer**

Circuit 1: Pull-down resistor

Create a circuit that uses a pushbutton to activate a piezo buzzer. You should use one of the digital inputs on the Arduino along with a pull-down resistor. The voltage sent to the buzzer should come from one of the analog outputs (digital PWM pins). You can look at the example code for making tones on a piezo buzzer at the end of this document.

Task: Build the circuit as described above.
Task: Draw the schematic of this circuit.
Task: Write code that make the piezo buzzer buzz when you press the button once and turn off the second time you push the button.

Circuit 2: Sensor instrument

Create an "instrument" that can be played using an LDR (Light Dependant Resistor). The frequency of the playback tone needs to be controlled by the amount of light hitting the LDR. The LDR must be connected to an analog input pin on the Arduino and use a pull-up resistor. You must also use the **play_tone** function provided below to generate the sound. This requires using digital output. The mapping between light/frequency is up to you. You will likely need to use the serial monitor to see what kind of input you are getting each sensor and use the map() function to adjust it.

Task: Build the circuit as described above. Add at least one decoupling capacitor to your circuit (10 uF).
Task: Write the code that allows the buzzer to behave as we want. Don't forget to read out the light sensor with the serial monitor.
Optional task: replace the pull-up resistor with the internal pull-up resistor of Arduino (activated through code).

Arduino Code for playing a tone on a piezo puzzer

Based on http://www.arduino.cc/en/Tutorial/PlayMelody

Tones are created by quickly pulsing a speaker on and off using PWM, to create signature frequencies. Each note has a frequency, created by varying the period of vibration, measured in microseconds. We'll use pulse-width modulation (PWM) to create that vibration.

We calculate the pulse-width to be half the period; we pulse the speaker HIGH for 'pulse-width' microseconds, then LOW for 'pulse-width' microseconds. This pulsing creates a vibration of the desired frequency. Modifications to code by Aaron Levisohn. Based on original code by (cleft) 2005 D. Cuartielles for K3

Code:

```
//TONES  ==========================================

// Start by defining the relationship between note, period, & frequency.

* c 3830 // 261 Hz
* d 3400 // 294 Hz
* e 3038 // 329 Hz
* f 2864 // 349 Hz
* g 2550 // 392 Hz
* a 2272 // 440 Hz
* b 2028 // 493 Hz
* C 1912 // 523 Hz
*/

// SETUP ============================================
// Set up speaker on a PWM pin (digital 9)


int speakerOut = 9;

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

// PLAY TONE  ==============================================
// Pulse the speaker to play a tone for a particular duration
void playTone(long duration, int tone) {
	long elapsed_time = 0;
	while (elapsed_time < duration) {

		digitalWrite(speakerOut,HIGH);
		delayMicroseconds(tone / 2);

		 // DOWN
		digitalWrite(speakerOut, LOW);
		delayMicroseconds(tone / 2);

		 // Keep track of how long we pulsed
		elapsed_time += (tone);
	}
}

void loop() {

	playTone(10000, 3830); //playTone(duration, tone)

}
```

Arduino Map Function

The map function re-maps a number from one range to another. That is, a value of **fromLow** would get mapped to **toLow**, a value of **fromHigh** to **toHigh**, values in-between to values in-between, etc.

```
y = map(x, 0, 50, 0, 1024);
```

Maps an input number (x) that lies between 0 and 255 to a proportional value between 0 and 1024.

Code:

```
void loop() {
	int val = analogRead(0);
	val = map(val, 0, 255, 0, 1024);
	analogWrite(9, val);
}
```

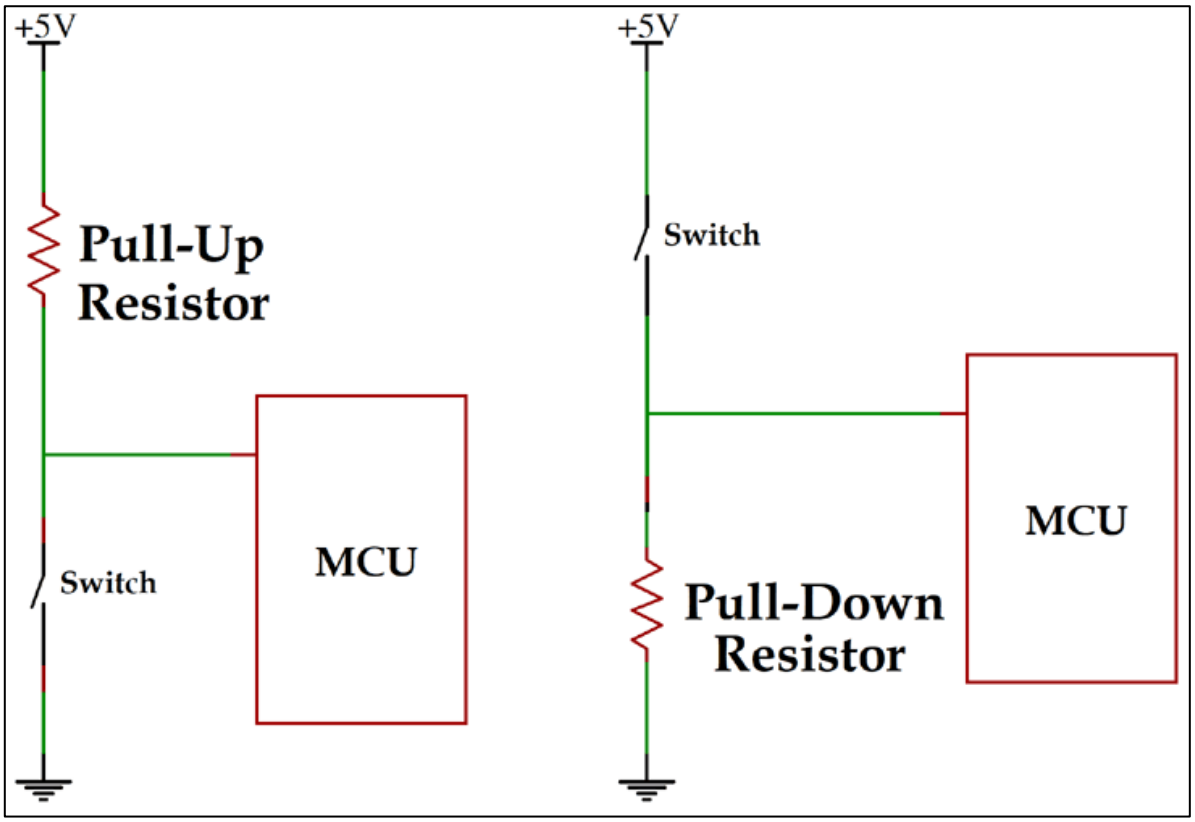
http://www.arduino.cc/en/Reference/Map

Pull-down resistor

A pull-down resistors is used to ensure that there is a full voltage drop to 0v when a switch is flipped. This is essential for the functioning of a switch connected to one of the digital input pins on the Arduino.

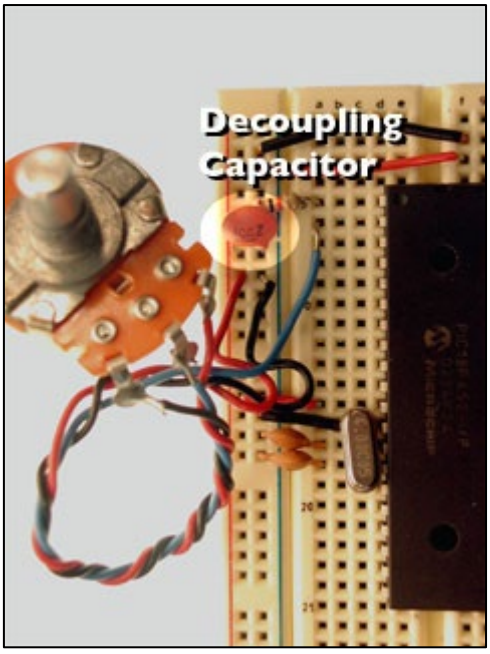
A resistor connected to the ground is a pull-down resistor. A resistor connected to Vcc is pull-up resistor.

Pull-down resistors also prevent short circuits between the ground and the input pin on the microcontroller. Without this precaution the chip could be damaged. More information can be found here: https://docs.arduino.cc/built-in-examples/digital/Button



Decoupling Analog Inputs

If you find the readings from your analog inputs are inconsistent (for example, you see changes on one channel when the sensor on a different channel is the one sensing action), it helps to decouple your input circuit. Decoupling means smoothing out the dips and spikes going into the circuit from the rest of your microcontroller circuit. To do this, place a 0.1microfarad capacitor from voltage to ground as close to where the analog input connects to voltage as in the figure below.



A capacitor used this way is referred to as a decoupling capacitor. You'll see them a lot in electronic circuits. Think of them as tiny surge protectors.

More information: https://learn.sparkfun.com/tutorials/capacitors/application-examples