

# Making Noise: Sound Art and Digital Media

UGS/CS 2050, Spring 2016



0	SoundCloud account	5
1	Induction Coil Field Recordings	7
2	Soldering Practice	14
3	Arduino Sound Program - Part I	27
4	Arduino Sound Program - Part II	38
5	Hardware Hacking	40
6	Oscillators	72
7	Final Sound Art Project	82



# 0 SoundCloud Account

Throughout this course you will be asked to share/submit recorded sound bites. We would like for you to use SoundCloud to store and manage your recordings. You may then link to your SoundCloud page when submitting assignments to Canvas.

For this assignment, please simply create a SoundCloud account, if you do not already have one.

We have also created a SoundCloud group associated with the class:

<https://soundcloud.com/groups/ugs-2050>

You should join this group once you've made your account. This will be a place where we can share sounds with the rest of the class.

You should upload one or two sounds to your SoundCloud account to make sure everything is functioning, and that you have things set up so that Nina and I can see your account. You can upload a sound clip of your own, or you can choose a couple of the silly sound effects posted on canvas alongside the assignment to upload.

## ***What to turn in...***

Upload your recording to SoundCloud and submit a link to your SoundCloud account on Canvas.



# 1 INDUCTION COIL FIELD RECORDINGS

In this lab, you will use an induction coil pickup and a recorder/amplifier to scout for interesting electromagnetic sounds.

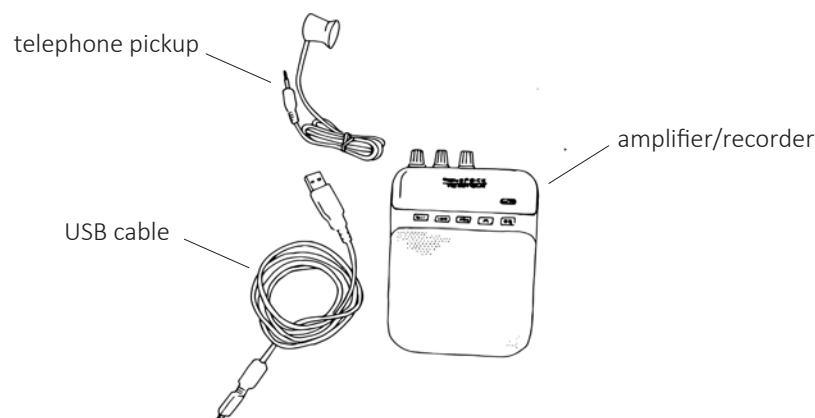
**Reading:** Before you begin, please review Chapter 3 in the textbook.

## Lab apparatus:

The lab apparatus includes a amplifier/recorder, an induction coil (a.k.a. telephone pickup), and a USB cable for transferring your data to a machine. The complete usage instructions are provided on page 9.

To listen to/record electromagnetic sounds:

1. Plug your telephone pickup into the Mic Jack.
2. Turn the On/Off knob (this also functions as a volume control).
3. Press the mond (<M>) button until you reach "Guitar REC" mode. At this point you should be able to hear any E&M sounds being picked up by the telephone pickup.
4. To record, press the <play/pause> button. Press again and hold for two seconds to stop recording.
5. To play back your recordings, use the mode button to get to "MP3 Play" mode.



## Collecting Samples:

Once you've set up your amplifier/recorder, take a walk around your house/lab/dorm room/neighborhood holding the induction coil up to various devices.

A few things to try:

- sniff around your laptop or desktop computer, or your cell phone
- if for some reason you have access to a landline, eavesdrop on a conversation
- sniff around the command interface of your microwave
- try sniffing the surface of electrical boxes on the sidewalk or near crosswalks.

Once you find an interesting sound, record it by hitting the "play/pause" button the the record mode. Record at least five of your favorite electromagnetic sounds. Each clip should be approx. 15 seconds long. Annotate information about each sound such as:

- what it is
- where you collected it
- when you collected it
- what you think is going on
- any other notes, observations.
- feel free to make sketches, take photos, etc.

## Extremely simple use of Audacity to normalize sound clips

Audacity is a few open-source audio-editing package. It falls in the same category of applications as things like

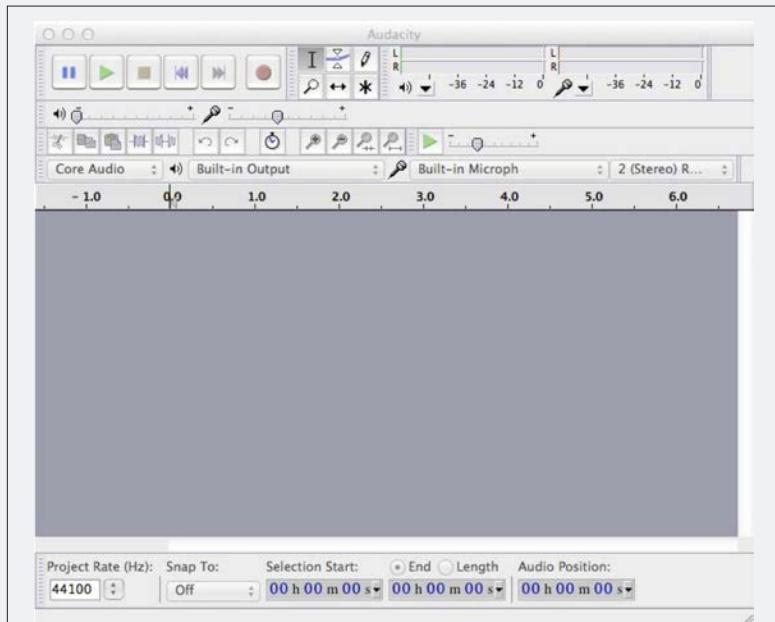
- Garage Band: <https://www.apple.com/mac/garageband/>
- WavePad: <http://www.nch.com.au/wavepad/>
- WaveShop: <http://waveshop.sourceforge.net/>
- Adobe Audition: <https://creative.adobe.com/products/audition?PID=7105813>
- Avid Pro Tools: <http://www.avid.com/US/products/family/Pro-Tools>
- Sony Sound Forge: <http://www.sonycreativesoftware.com/soundforgesoftware>

All of these audio editing tools have their pros and cons – and some are vastly more complex than we'll need. And some cost a LOT of money.

Audacity is free and open source, and it runs on Mac, Windows, and Linux. It also does everything we need it to do. It's also installed on the machines in our lab. I recommend you use it unless you already have and use some other tool.

You can download Audacity from:  
<http://audacity.sourceforge.net/>

When you start up Audacity it looks something like this:



In the upper left you see the pause, play, stop, go to beginning, go to end, and record buttons that you might have expected for a sound-editing program. The L and R bars on the right top of the screen are the level indicators for playback and for recording. At the bottom left there is an indicator of the project sample rate in Hertz (Hz). One Hertz is one cycle (one up and down of the signal) per second. 100 Hz is 100 cycles (ups and downs of the signal) per second. The Project Rate is an indication of the rate at which digital samples are made of the music you're recording or playing back. 44,100Hz means that the editor is assuming that there are 44,100 samples of the audio waveform made every second.

That sounds like a lot, but it's not really all that fast for a computer. Consider that your laptop is likely running at a rate of more than 2GHz (2,000,000,000Hz). That means that your laptop

is executing very basic machine-language instructions at a rate of 2 billion of those tiny instructions per second. That means there are a LOT of computer instructions that can be done between every music sample.

In practice, the human range of hearing is roughly 20Hz to 20,000Hz (although old guys like me have lost a lot of high-frequency range to my hearing). The default sampling rate of most "high quality" audio files is 44,100 Hz. The reason has to do with something called the "Nyquist-Shannon sampling theorem," or sometimes just called the 'Nyquist sampling rate.' According to this theorem, if you want to accurately reproduce signals at a particular frequency (1350Hz, for example), you need to sample that signal at a rate that is at least 2x higher than the rate you're sampling (e.g. 2700Hz sampling rate in this example).

So, if you want to accurately represent signals up to the top end of human hearing (20,000Hz), you need to sample at least twice as fast (40,000Hz). When they decided how to format sound on CDs, they chose 44,100Hz rate to make sure that they could reproduce sounds that humans cared about. Here we could go into a long discussion of whether CDs really do represent correctly all the sounds people can hear, whether vinyl records are better, and whether CDs should really have used a 96,000Hz sampling rate. But we'll save that for in-class discussion.



When you've recorded some EM "sounds" on your amplifier/recorder, the file will be an mp3 file. As another aside, mp3 files are heavily compressed audio files to make them smaller than what you might fit on a CD. Again, we can argue in class about whether mp3s sound as good as a CD or not (spoiler – they generally don't sound as good as CDs, but you may not notice in the typical mp3 listening environment).

To transfer your recordings onto your machine, or to a lab machine, connect the amplifier to the machine using the provided USB cable. Turn the amp on, and the disk image should appear on the machine. Open the disk image, likely called "No NAME," and drag the "amp" directory onto the desktop. If there are many recordings in the directory, yours should be the most recent.

Once you've transferred your sound capture files to your machine, or to a lab machine, you can open them in Audacity and see what they look and sound like. Here's an example of a simple recording I made on the amplifier/recorder and loaded on my laptop. I used File -> Open in Audacity to open the rec00001.mp3 file that I got from the recorder's flash card.

What you see here is a stereo signal with the left channel on the top and right channel on the bottom. This is a bit of a lie. I used the inductive coil to do the recording, so it only recorded one signal. But it recorded that signal to both the left and right channel according to the mp3 file.

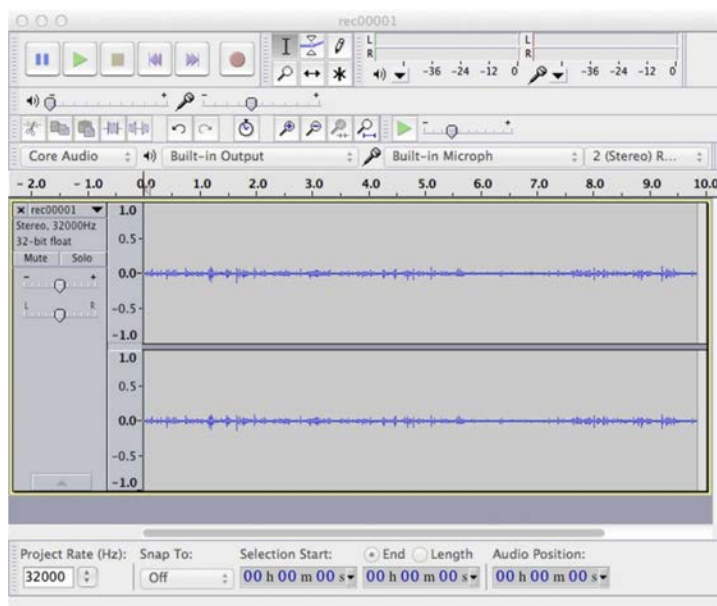


fig 1

By the way – note that the Project Rate has gone down to 32,000Hz. This is because the amplifier/recorder was set to a quality setting that used a lower sampling rate to make the files smaller. It's still completely fast enough to get good-sounding EM sounds.

So, the first thing I can do is get rid of that fake stereo issue. I used Tracks -> Stereo Track to Mono and collapse the signal to a single mono signal (fig 2).

Now you can play your signal using the green play button and see what it sounds like. It's probably pretty soft if it looks like this image. What you're seeing is a visual representation of the waveform. Time goes from left to right, and the loudness of the signal (the amplitude) is shown on the vertical axis. The audio is scaled so that the loudest sound possible goes from the bottom of the graph (-1.0) to the top of the graph (1.0). In this image the sound is pretty faint – it only goes up to a tiny bit above the 0 point.

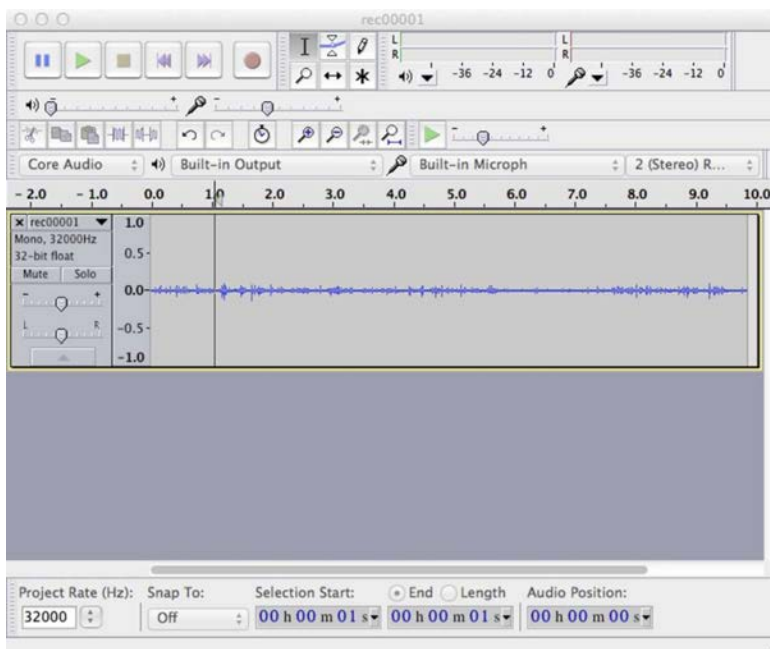


fig 2

If I use the zoom buttons (the ones that look like little magnifying glasses) I can see more detail on the audio signal. If I zoom WAY in to that little lump just to the right of the cursor in the previous figure, here's what I see (fig 3). It's a little wiggle of the audio signal.

That's pretty quiet!

One way to make it louder is to select the track and adjust the volume with Effect -> Amplify. You could, for example, select the entire clip (double click in the clip), and then use the Amplify effect to make things louder. In general, the Effect menu choice has a huge number of audio effects you can apply to your tracks. We won't use most of them, but they're fun to play around with.

One problem with "Amplify" is that you might amplify too much. Especially with digital audio, there is a maximum signal that can be represented. If you try to make it louder than the loudest that the digital representation will allow, it will just stop at the biggest value and stick there. Think about it this way – if the loudest number you can represent in the digital format is 32,768 (to pick a (sort of) random value), and you try to amplify things so that the loudest part of the signal is 50,000, then everything from 32,768 to 50,000 would be stuck at 32,768. This is called "clipping" because the signal, which should be round on the top, is clipped to be flat on the top. This turns out to sound terrible! This is what it looks like to vastly over-amplify a signal:

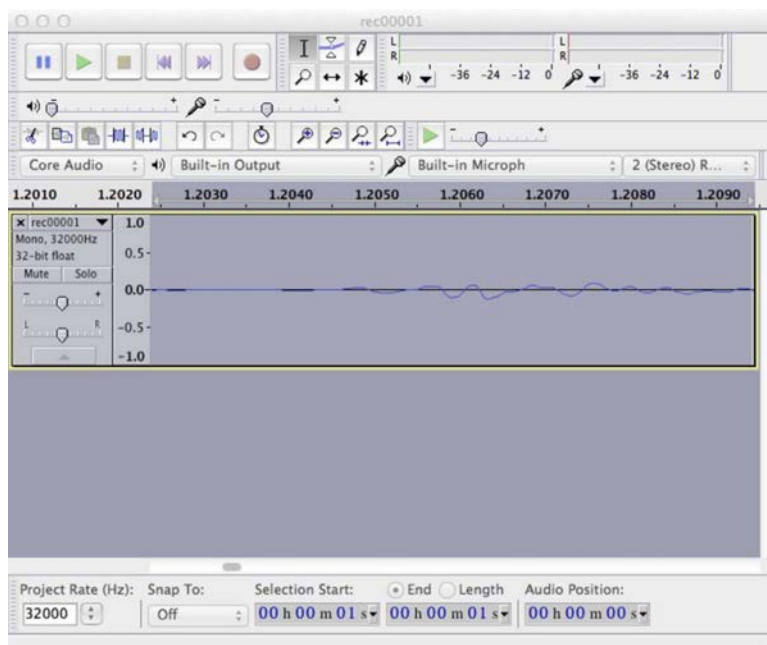


fig 3

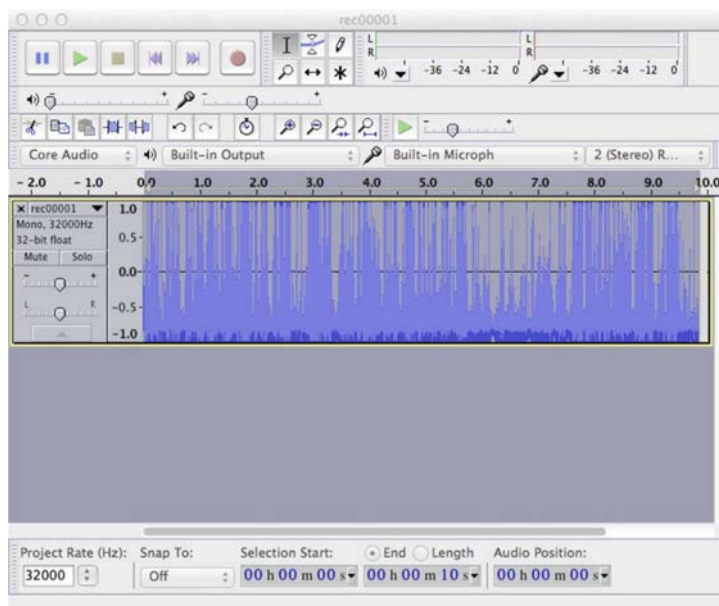
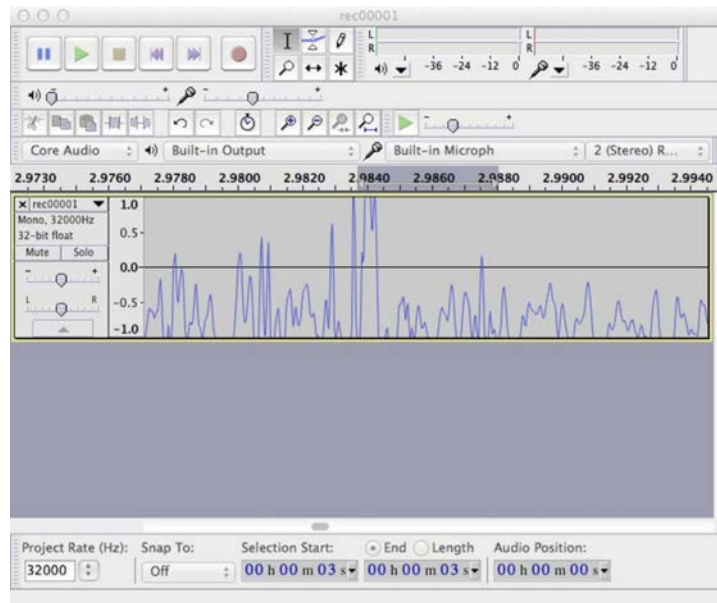
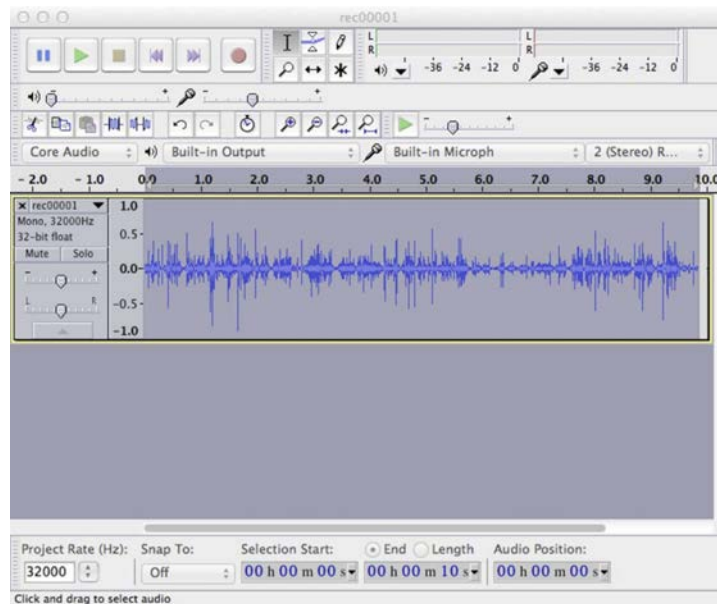


fig 4

If I zoom in again you can see the clipping. It's a little hard to see until you know what you're looking at, but all those flat bottoms on the waveforms are clipped, and will sound awful.

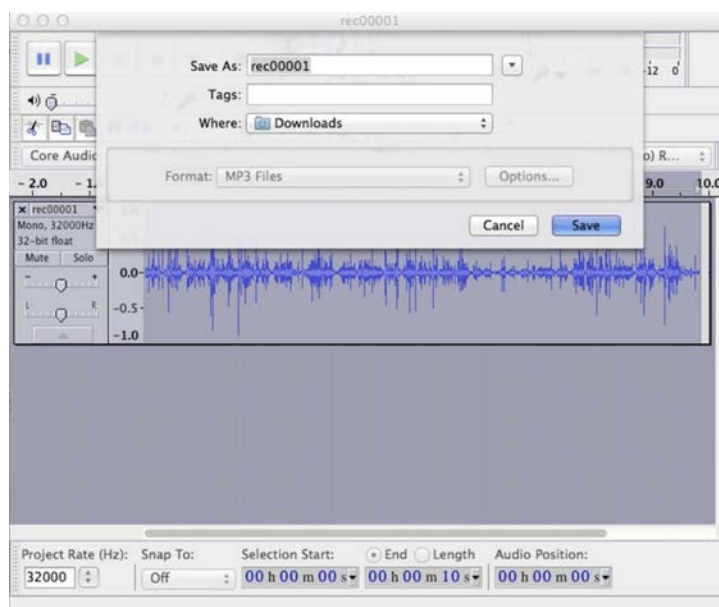


So, we'd like to avoid that if possible, but also amplify the weak signal so it's louder. This is where Effect -> Normalize comes in. This is an effect that looks at the highest and lowest parts of the audio file, and then adjusts the volume of the whole clip so that the highest and lowest are at a nice loud volume, and that no clipping happens. Here's the same starting waveform but with the Normalize Effect added to the entire clip. Use the default settings of "remove DC offset" and "max value -1.0db."



Now – this is more like it! The waveform has been amplified, but amplified just enough so it's as loud as it can get without clipping. This will get the most out of your recorded signals in terms of hearing everything that's in the signal.

After you perform the Normalization, you can output the sound as either a .wav file (high quality, but big files) or as an mp3 (highly compressed, and absolutely fine for our purposes). Use File -> Export Audio to save your file. Select mp3 to save space, and you don't need to mess with the options unless you want to. You can fill in the meta-data during the mp3 writing process if you want to, but its not required.



I'd like you to normalize all your inductive field recordings. Here's the cheat-sheet version of the process:

1. Download your sound files from your amplifier/recorder
2. Start Audacity
3. Read your mp3 files with File -> Open
4. Select the entire track with a double-click
5. Use Tracks-> Stereo to Mono to get it back to a single sound track
6. Use Effect -> Normalize to normalize the volume of the track
7. Write the new normalized mp3 using File -> Export Audio

The normalized recordings are what you should turn in.

We will use other effects and features of Audacity later, but this is the important part for now...

### ***What to turn in...***

- Recording of at least five of your favorite electromagnetic sounds, uploaded to SoundCloud (provide links in Canvas) Each clip should be approx. 15 seconds long. Annotate information about each sound such as:
  - what it is
  - where you collected it
  - when you collected it
  - what you think is going on
  - any other notes, observations, etc. including sketches, photos, and other documentation about the location and circumstances of the collection.

MONOPRICE  
PRO AUDIO SERIES

# 5-watt guitar amplifier

Portable Recorder, and USB Audio Interface

611700

Thank you for purchasing the MONOPRICE® Guitar Amplifier and Recorder! This device features:

- 5-watt mono amplifier driving a single 4-ohm speaker
- Guitar distortion effects
- 1/4" TS guitar input jack
- The ability to play mp3, wav, and wma files stored on a micro SD card
- The ability to record 128 Kbps mp3s to a micro SD card
- 3.5mm microphone input jack
- 3.5mm auxiliary input jack
- 3.5mm headphone output jack

## PACKAGE CONTENTS

After receiving the product, please inventory the contents to ensure you have all the proper parts, as listed below. If anything is missing or damaged, please contact Monoprice Customer Service for a replacement.

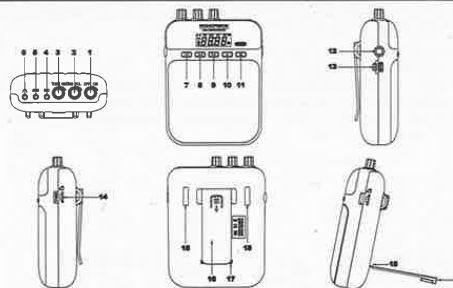
- 1x 5-watt Guitar Amplifier
- 1x Mini USB Cable
- 1x User's Manual

## SAFETY GUIDELINES

For best results, please read this manual, paying extra attention to these safety guidelines. Keep this manual in a safe place for future reference.

- Do not expose this device to water or liquid of any kind.
- Do not expose this device to fire or extreme heat.
- Do not attempt to repair this device. There are no user serviceable parts.
- This device contains a rechargeable lithium battery. Please follow all local environmental disposal laws when discarding this device.
- Do not use alcohol, chemicals, or solvents to clean the outside of this device. Use only a soft, dry cloth to clean the unit.
- Do not continue to use this device after the low battery warning indicator begins to flash. Doing so can reduce the battery's storage capacity and overall usable life.
- Do not drop or throw the amp. Do not jump up and down on the amp with hobnailed boots. Doing so could damage the amp and render it inoperable.

## CONNECTIONS AND CONTROLS



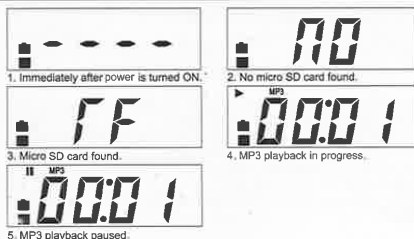
1. **ON/OFF Knob:** Turns the amplifier ON/OFF and controls the music playback volume level.
2. **VOL Knob:** Controls the guitar volume (Guitar mode only).
3. **TONE Knob:** Controls the amount of distortion effect to apply to the guitar signal (Guitar mode only).
4. **MIC Jack:** 3.5mm microphone input jack.
5. **AUX Jack:** 3.5mm audio input jack. The amp automatically switches to GUITAR+AUX mode whenever a plug is inserted into the Aux jack.
6. **Headphone Jack:** 3.5mm mono audio output jack. The speaker automatically mutes when a plug is inserted into the headphone jack.
7. **<play/pause> Button:** Performs the play/pause function in MP3 and USB-SPEAKER modes. Mutes the speaker output in GUITAR mode. Starts and pauses recording in RECORD mode. Press and hold for 2 seconds to stop recording.
8. **<previous> Button:** Press once in MP3 or USB-SPEAKER modes to skip back to the previous music track. Press and hold for 2 seconds in MP3 mode to reverse play at 10x normal speed.
9. **<next> Button:** Press once in MP3 or USB-SPEAKER modes to skip forward to the next music track. Press and hold for 2 seconds in MP3 mode to forward play at 10x normal speed.
10. **M Button:** Press the mode selection button to cycle through the available modes. The MP3 mode cannot be selected unless a micro SD card with playable mp3, wav, and/or wma files is inserted into the Micro SD card slot. The AUX mode cannot be selected unless a plug is inserted into the Aux jack. The RECORD mode cannot be selected unless a usable micro SD card is inserted into the Micro SD card slot.
11. **EQ Button:** Pressing the EQ button in MP3 mode cycles through the five preset equalizer settings: Standard (EQ0), Classic (EQ1), Pop (EQ2), Rock (EQ3), and Country (EQ4). Press and hold the EQ button for 2 seconds in Guitar mode to turn the distortion effect on/off. Press the EQ button in RECORD mode to mute/unmute the speaker.
12. **1/4" TS Guitar Cable Input Jack:** Plug your guitar cable into this jack.
13. **Mini USB Port:** Plug the amplifier into a USB charging source (computer, wall adapter, etc.) using the included mini USB cable to charge the built-in lithium battery.
14. **Micro SD (TF) Card Slot:** Accepts micro SD cards with capacities up to 32 GB. A micro SD card with playable mp3, wav, and/or wma files is required to enter MP3 mode. A micro SD card with available space is required to enter RECORD mode.
15. **Strap Loops:** Use these loops to attach a strap (not included).
16. **Belt Clip:** Use the belt clip to attach the amplifier to your belt for portable operation.
17. **Stand Slot:** The belt clip also doubles as a stand. Pull up on the small clip with the arrow then slide the clip downward to remove it from the amplifier body. The loosened clip can then be inserted into the slot on the back to prop the amp up, as shown in the image above.

## LED SCREEN



1. Playback and Recording time display.
2. Battery Charge Indicator  
Both halves of the battery indicator flash at the same time = Battery charge expended.  
The amp will power OFF within 10 seconds.  
The lower half of the battery flashes = Battery charge is about 50%.  
The two battery halves are both lit constantly = Battery charge is about 100%.  
The two battery halves flash in succession = The battery is being charged.
3. RECORD mode indicator.
4. Music Playback indicator.
5. Music Pause indicator.
6. MP3 mode indicator.
7. Guitar mode indicator.
8. AUX mode indicator.
9. USB-SPEAKER mode indicator.
10. Speaker Mute indicator.

## OPERATION



### 1. MP3 MODE

If a micro SD card with playable mp3, wav, and/or wma files is present when the amp is powered on, it will automatically enter MP3 mode and begin playback of the music files. If no card is present the LED screen displays "NO" (see Figure 2).

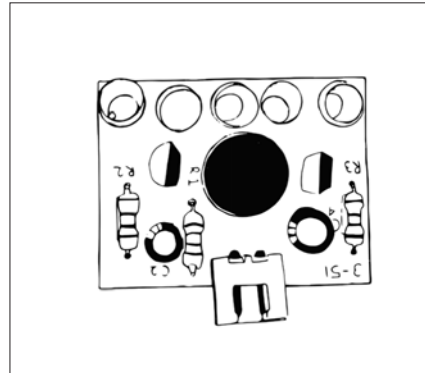
- a. Press the <previous> or <next> button to skip to the previous or next track.
- b. Press and hold the <previous> or <next> button to perform fast backwards or forwards playback at 10x normal speed.
- c. Press the <play/pause> button to pause music playback. Press it again to resume playback.
- d. Press the EQ button to cycle through the five equalizer presets: Standard (EQ0), Classic (EQ1), Pop (EQ2), Rock (EQ3), and Country (EQ4).
- e. Use the power knob to control playback volume.
- f. The other knobs have no effect in MP3 mode.
- g. The amp remembers which track and the track location of the last file played and will resume from that point when MP3 playback is initiated.



## 2 SOLDERING PRACTICE

For this project I'll hand out a small kit for an electronic flashing circuit. Assembling the kit involves soldering a set of through-hole components on a small circuit board. When assembled, the kit will flash a set of LEDs when it hears sounds above a certain volume. The main purpose of this assignment is to get some practice with soldering.

I'll hand out the kits in class.



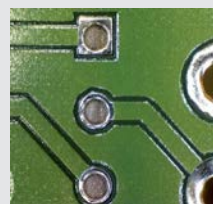
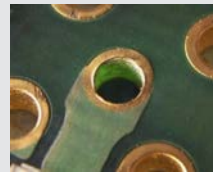
The following section provides a breif introduction to soldering and through-hole components.

### Soldering - basic instructions

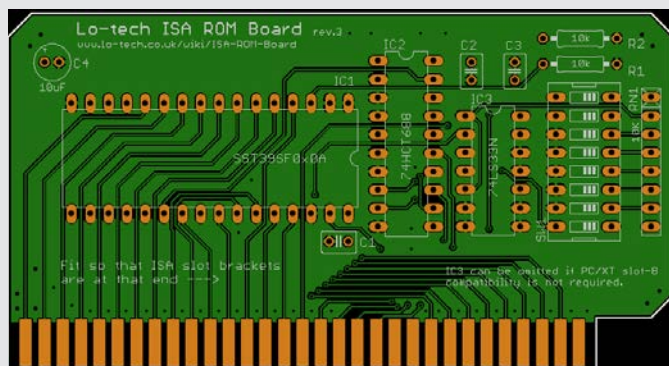
- Heat the material to be soldered
- Let the Solder melt and flow around that material
- At the same time...don't heat things too much or for too long.
- It takes a little practice to get things just right.

### Through-hole components

- Components designed to go through a hole... :-)
- Circuit boards have holes in them with little metal donuts around the holes
- Those metal donuts connect the hole with wires (traces) on the circuit board
- Through-hole components have leads that stick through those holes so you can solder them to the donuts.



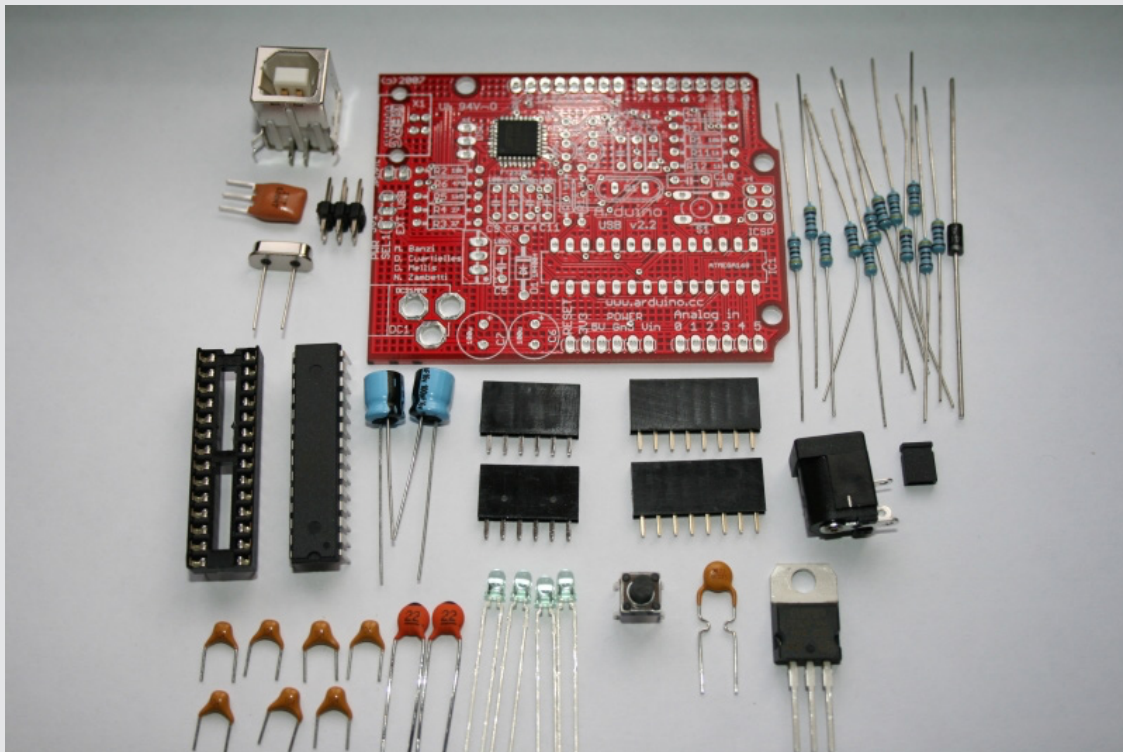
### Printed Circuit Board (PCB)



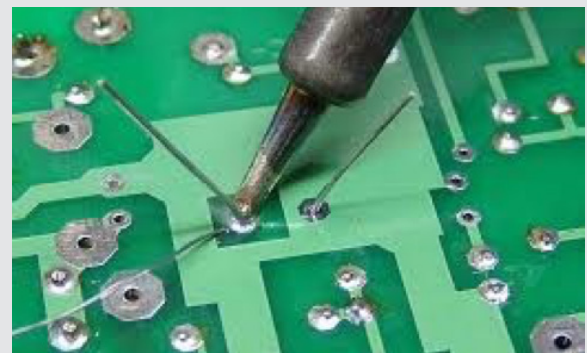
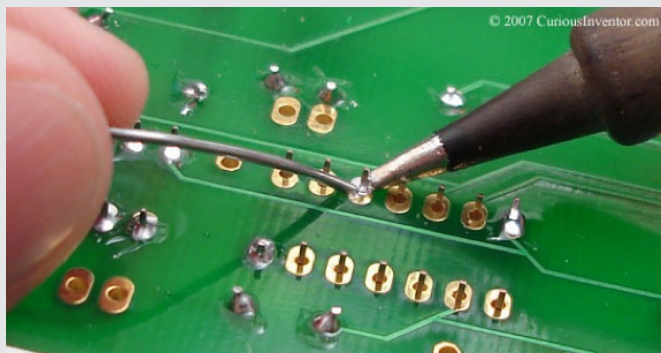
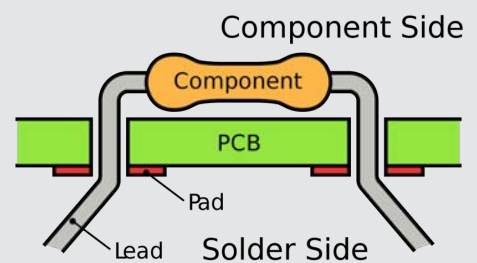
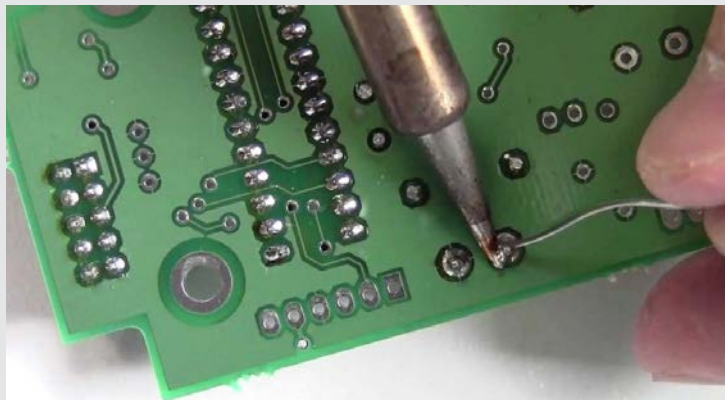
### PCB's with holes



## Through-hole components

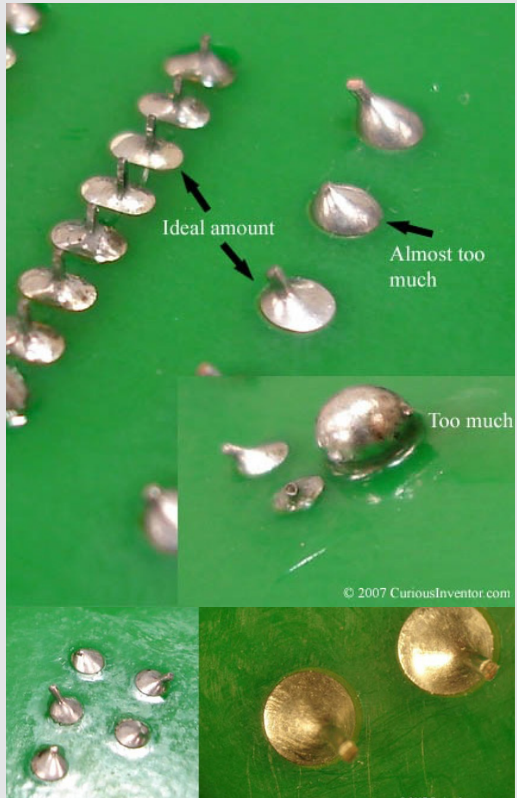


## Soldering through-hole components





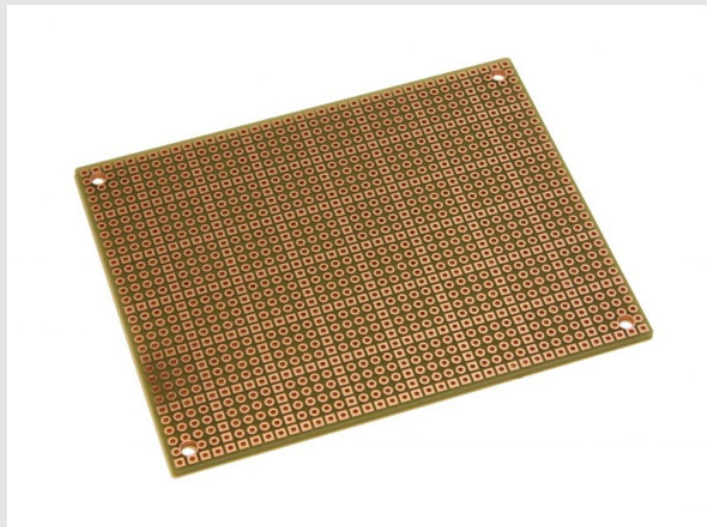
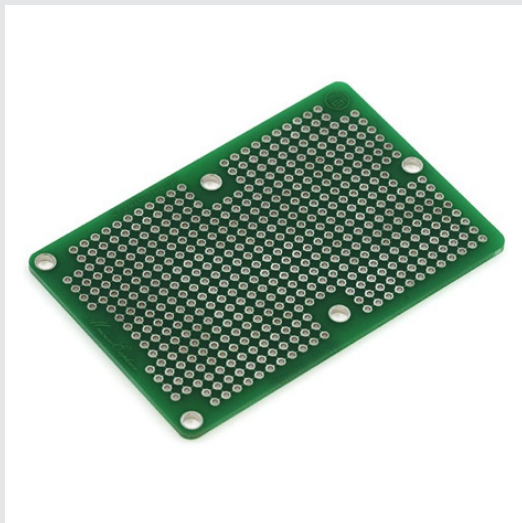
### Good through-hole joints



### Bad through-hole joints

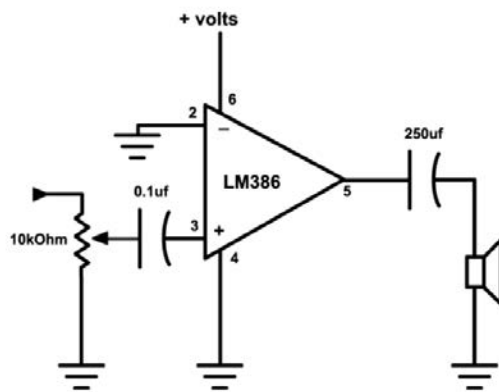


### Proto-boards

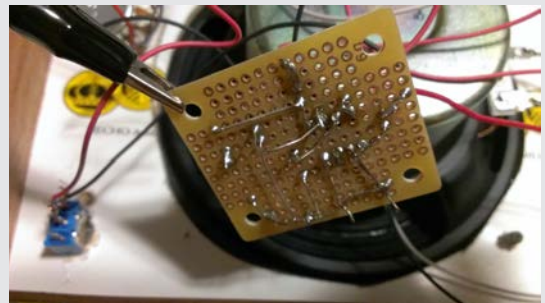
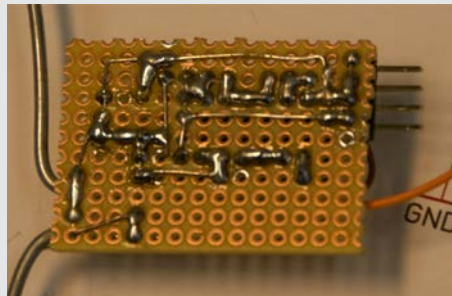
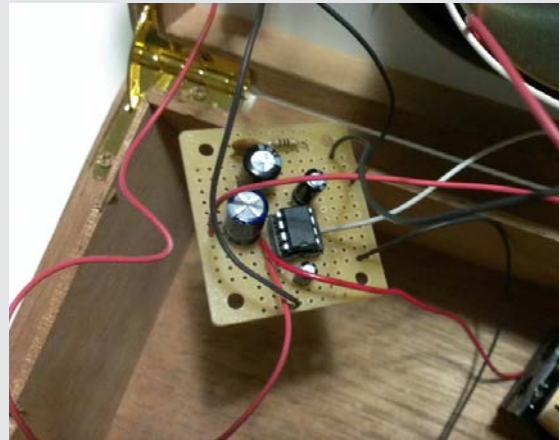




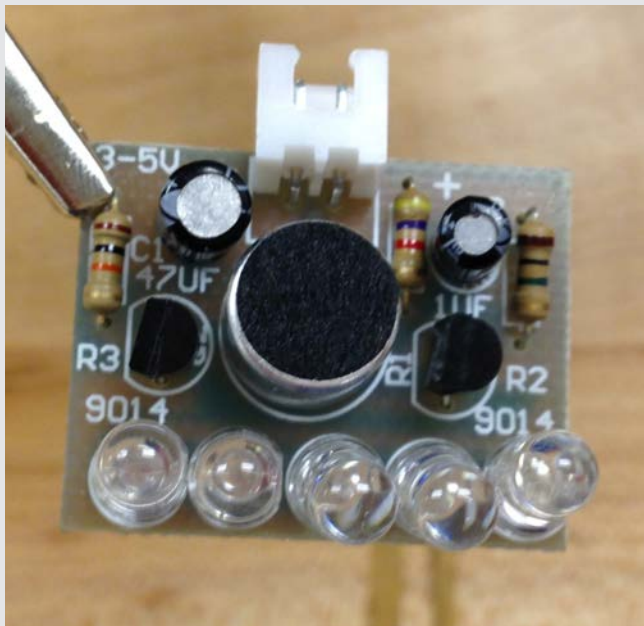
Amplifier schematic



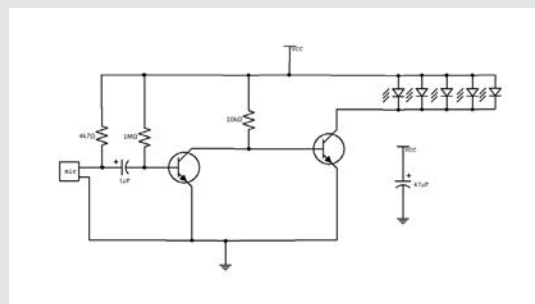
Proto-board amplifier



Light Flasher

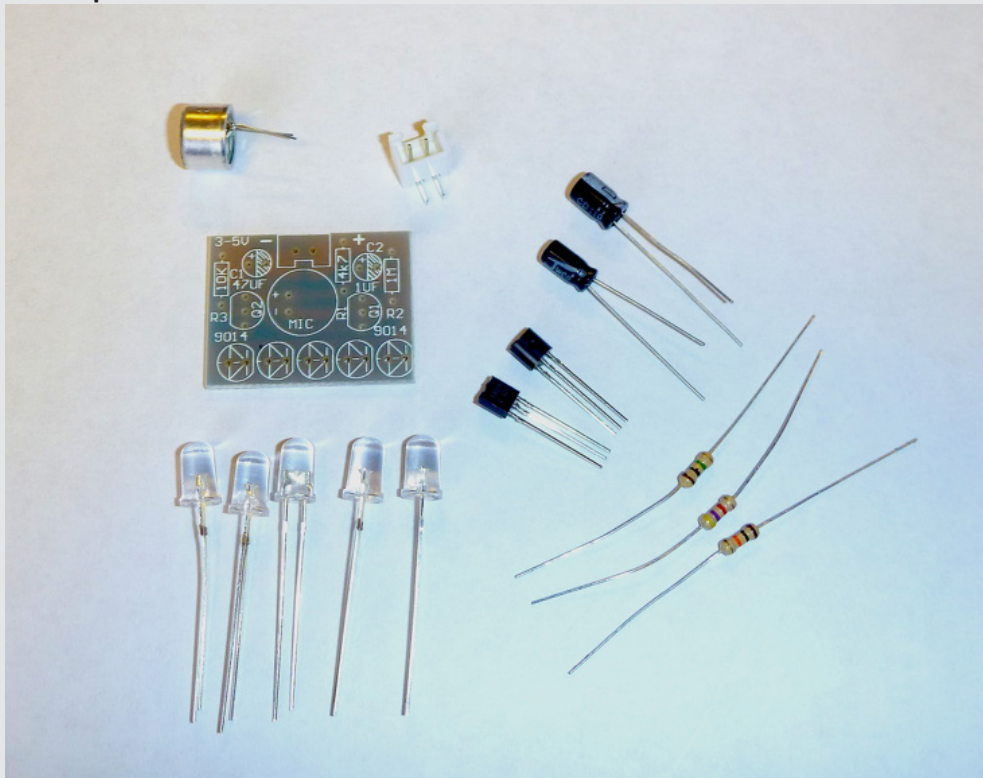


- LEDs will flash when you hold the mic up to noise/music



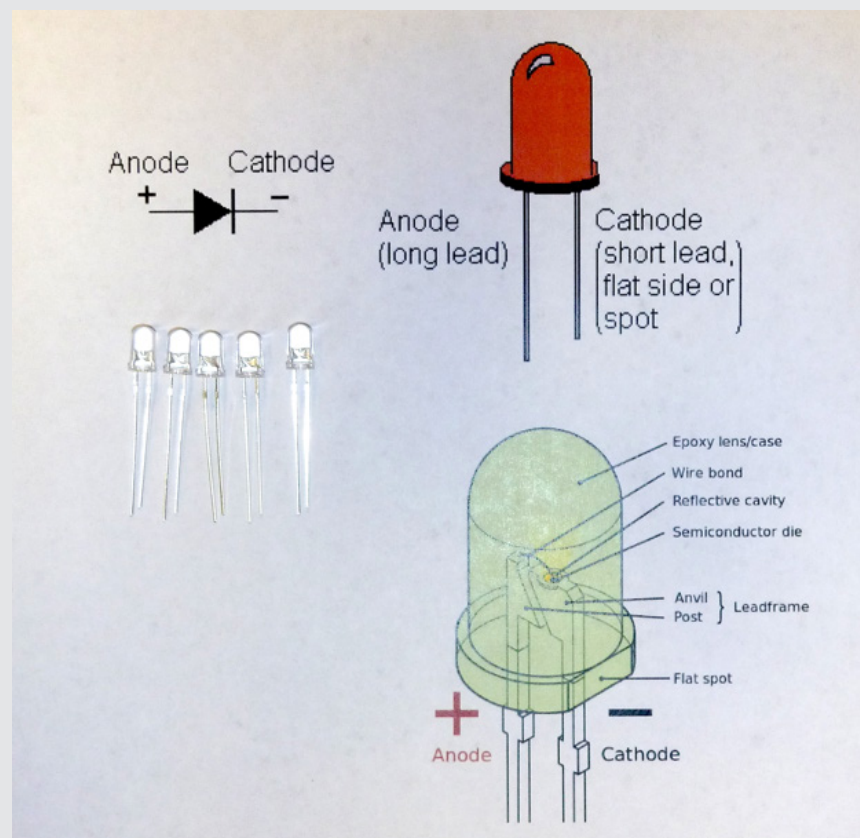
Circuit diagram

## Light Flasher - kit parts



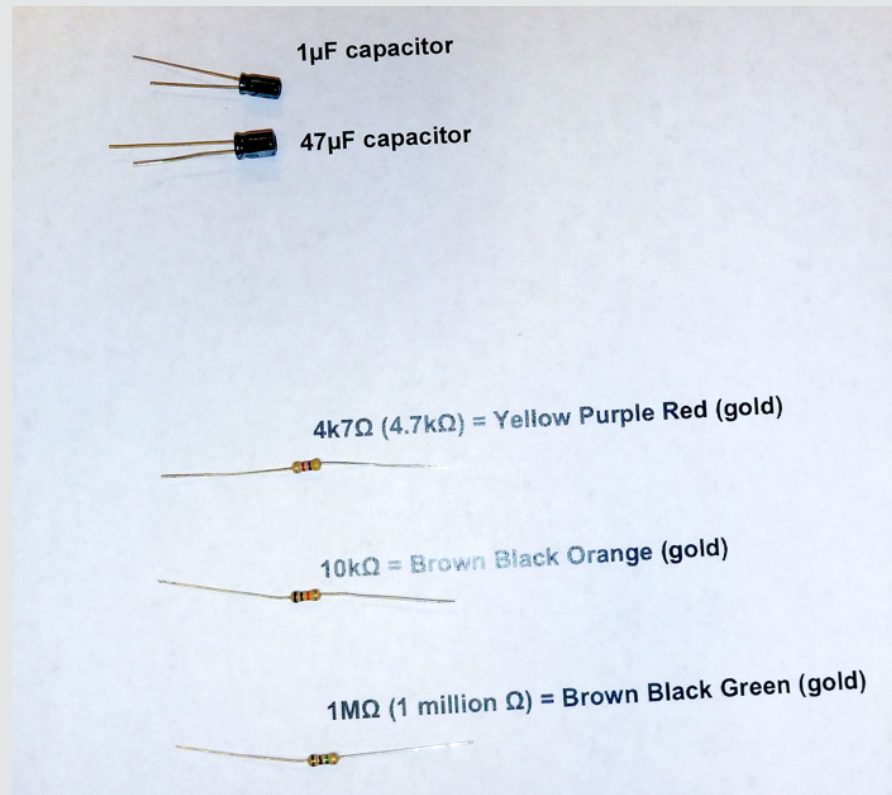
## Light Flasher - components

- LEDs have a direction.
- Anode is connected to the + side of the circuit (battery) and Cathode to the - side.
- They light up when current flows from + to -.



## Light Flasher - components

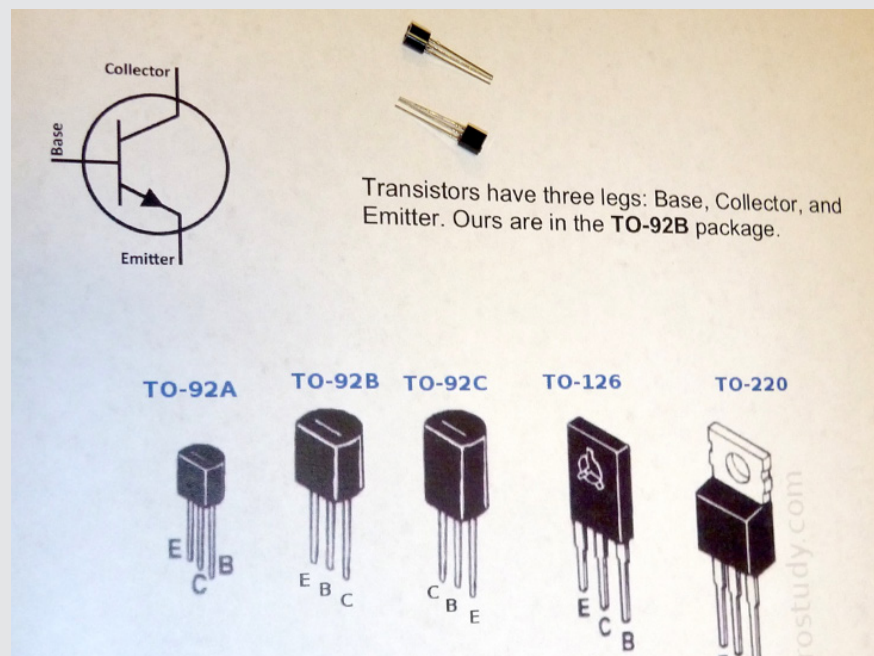
- Capacitors have their value printed on them



- Resistors have a color code to indicate their value

## Light Flasher - components

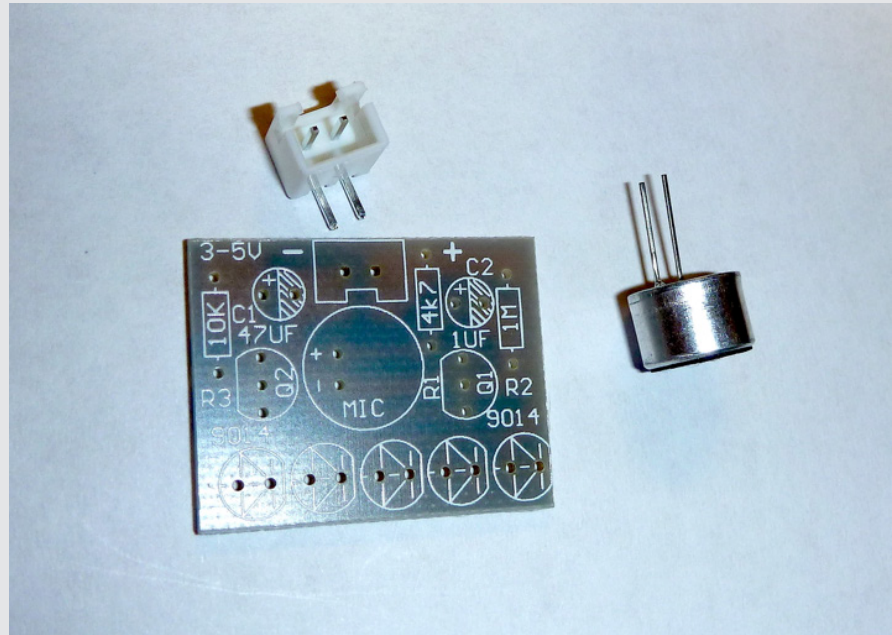
- Transistors have three connections. The flat side of the case orients the device





### Light Flasher - components

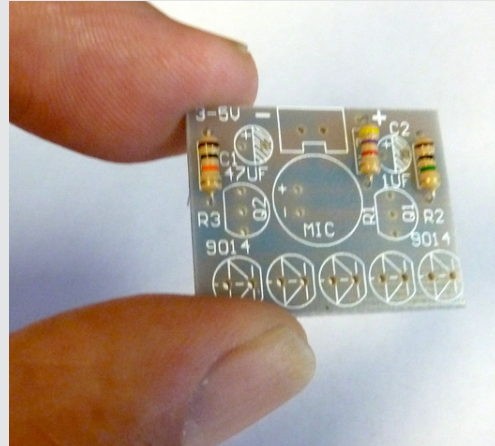
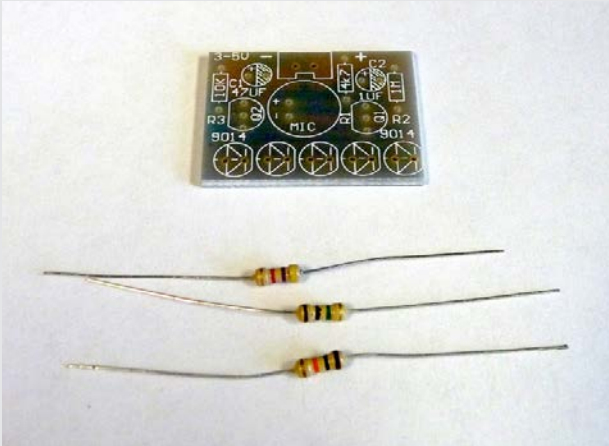
-The battery connector and the mic round out the components



Next we provide a step-by-step on the actual kit you'll be soldering. We'll also hand out some battery cases that you can use to power your circuit.

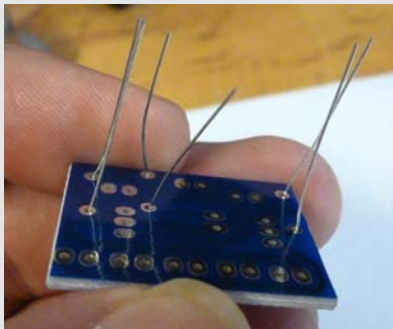
1

Start with the resistors. Bend the leads to fit through the holes

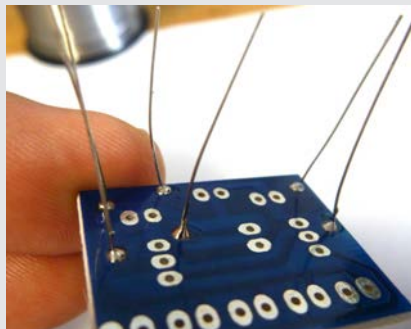


1

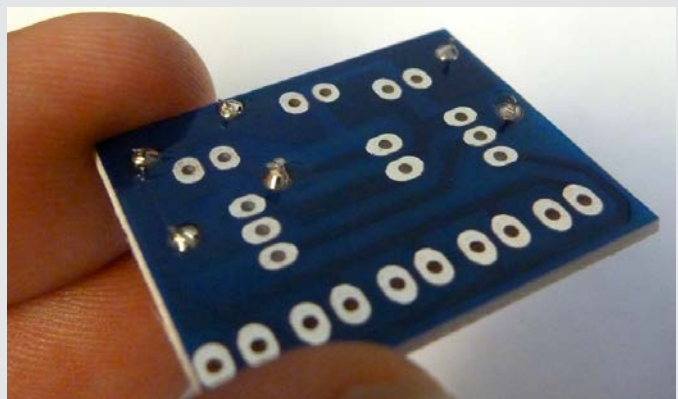
The leads poke through to the solder side of the board.



Solder 'em up

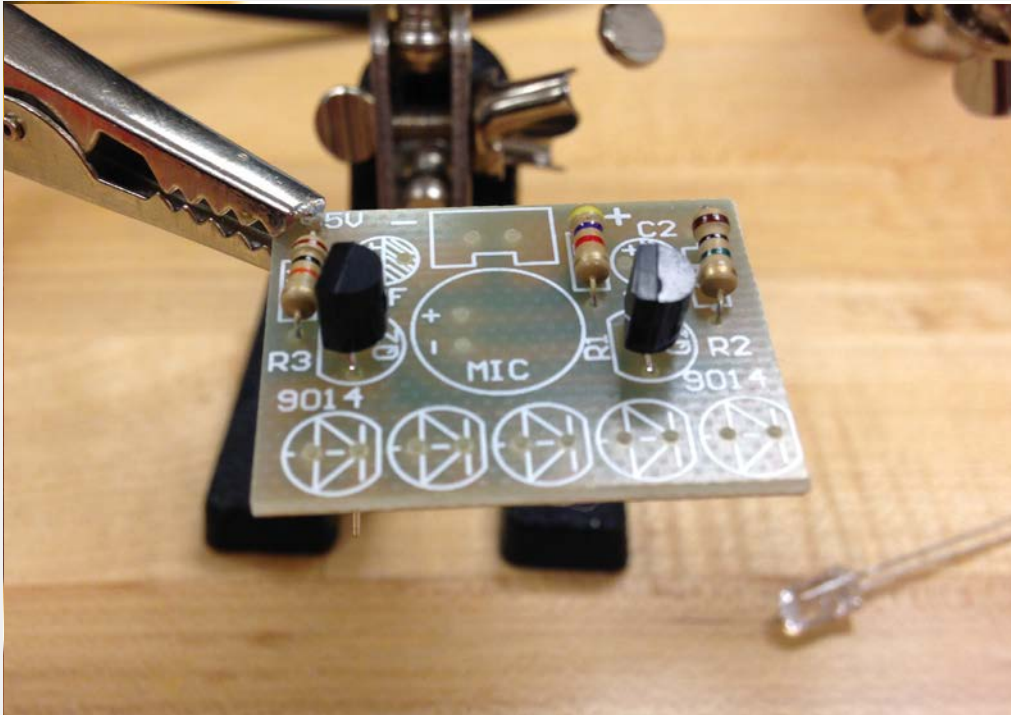


And then clip the excess leads!

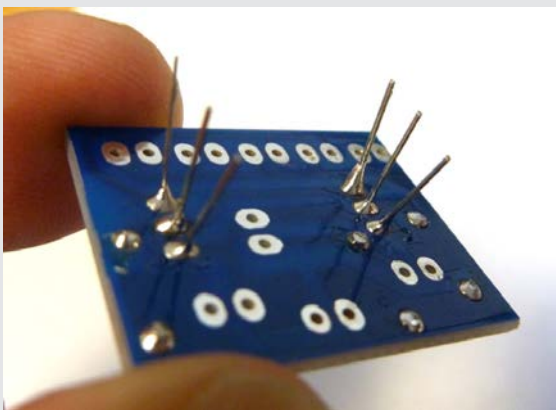


2

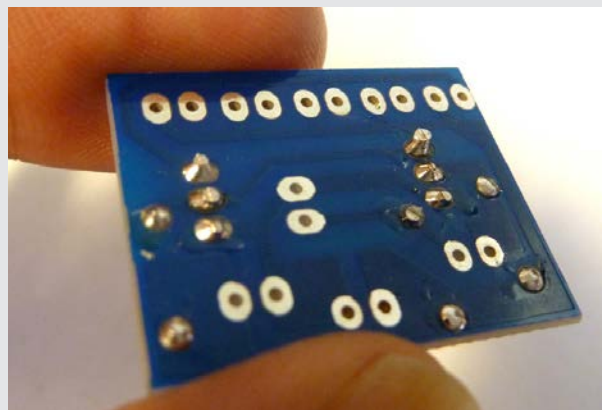
Now add the transistors. Use the flat side to orient them properly.



2



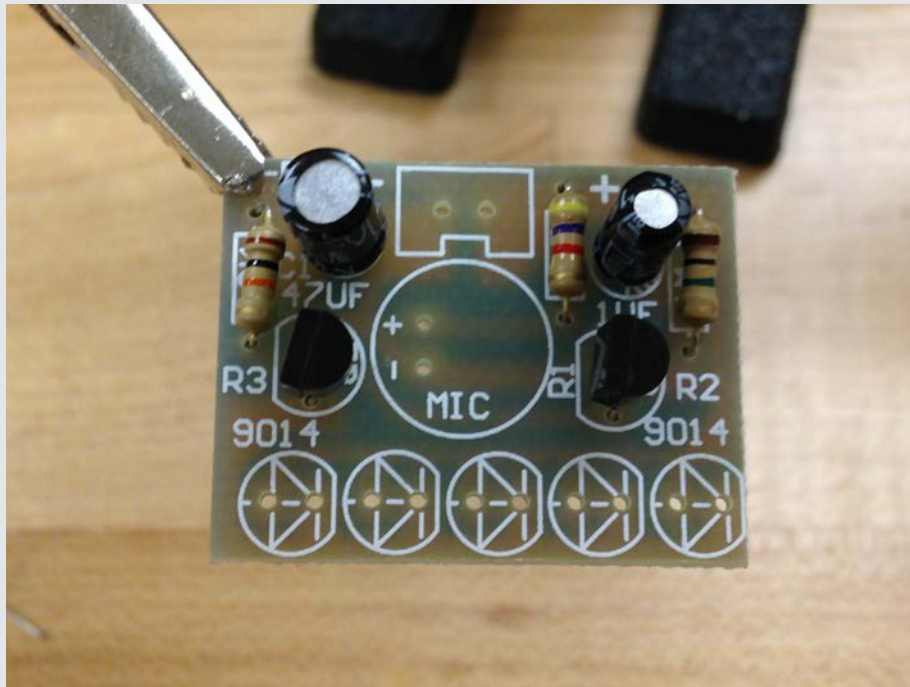
Apply solder...



and clip the leads

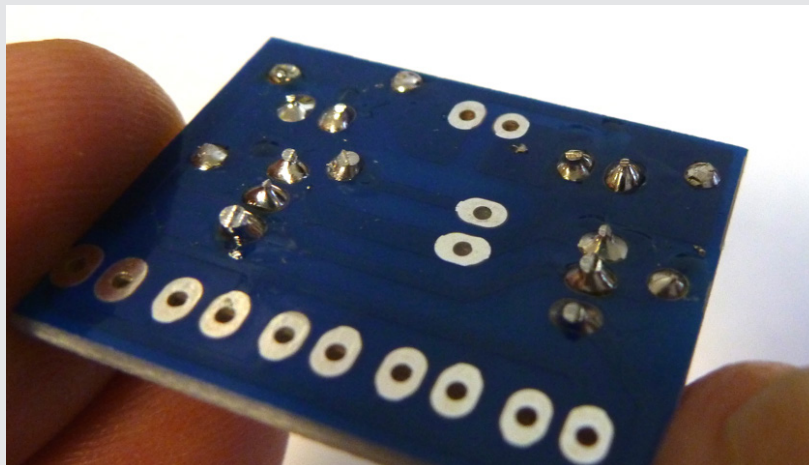
3

Now add the capacitors. Note that the white stripe is the - side of these caps. It's important to orient them correctly...



3

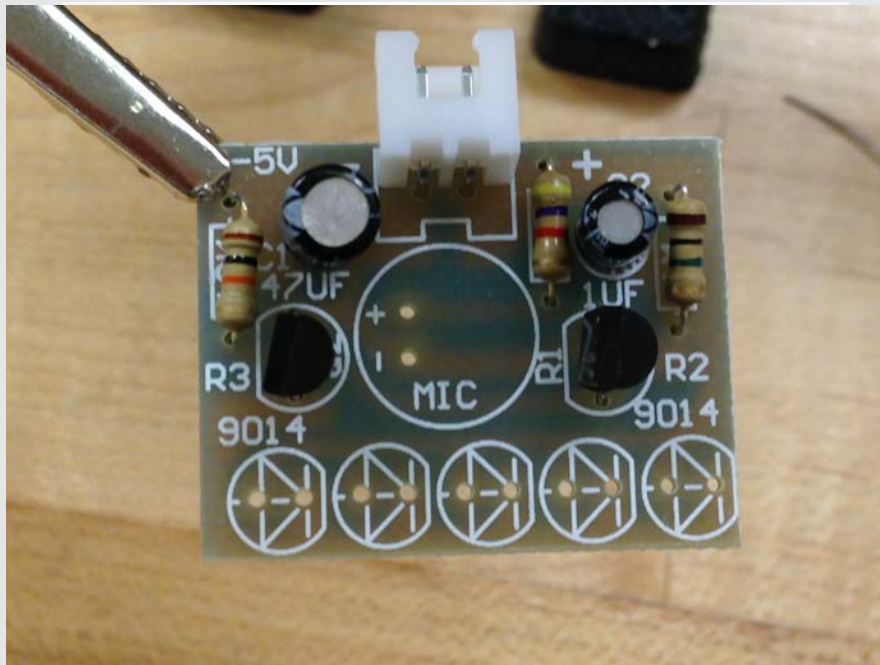
Solder and clip!





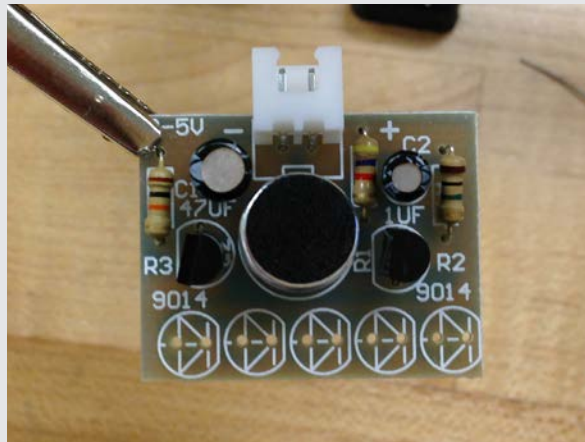
5

Solder in the battery connector

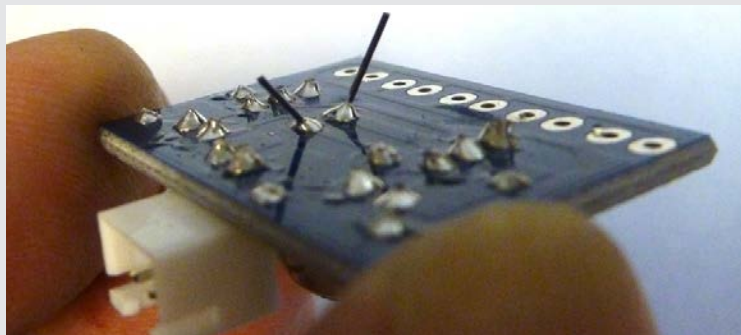


5

Now add the mic in the middle



Note the bent leads to help hold it in while soldering



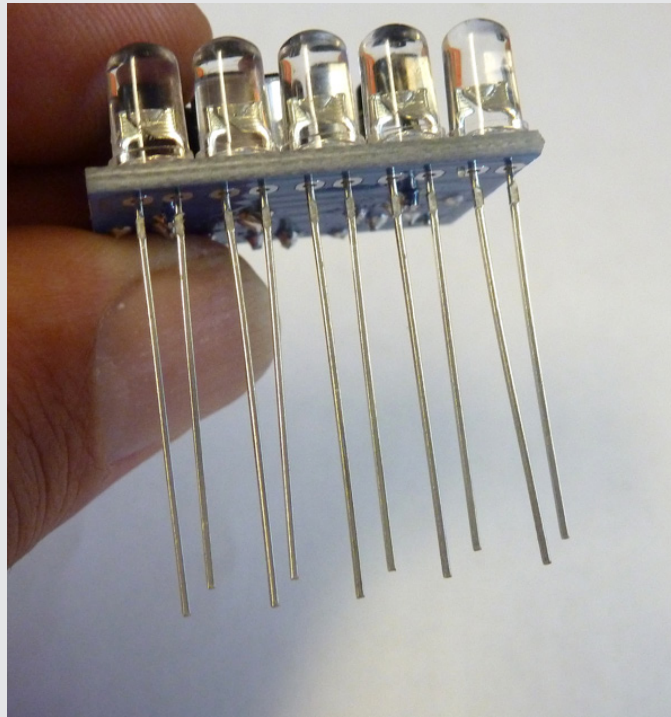


6

Finally add the LEDs.

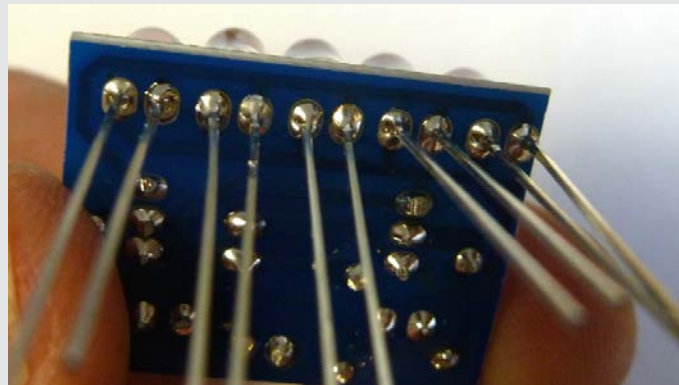
Make sure to get the direction correct. Remember that the Anode (+ input) is the longer lead.

You want current to flow from + (long) to - (short).

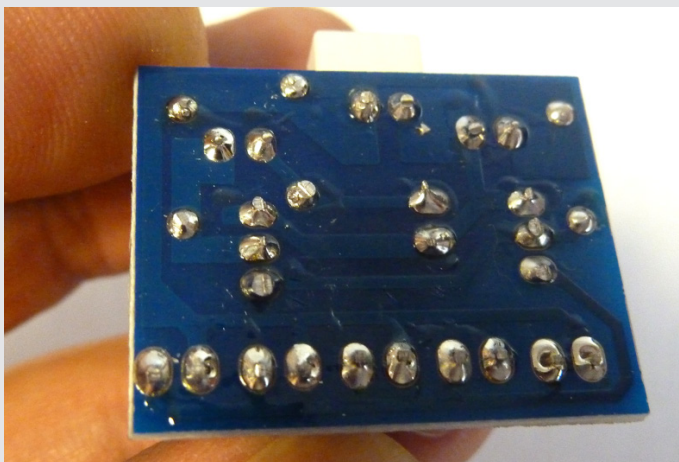


6

Solder...

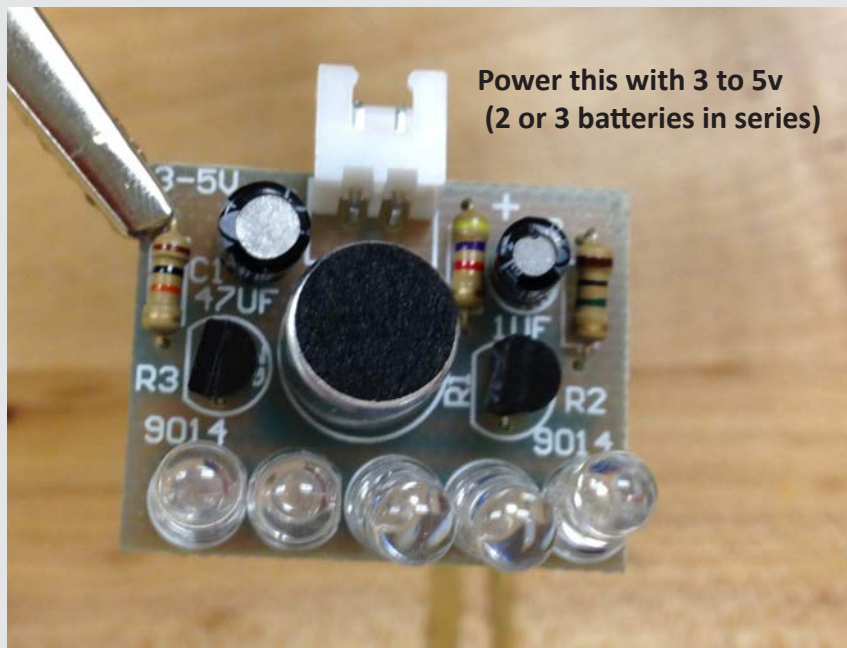


...and clip. And we're done!



7

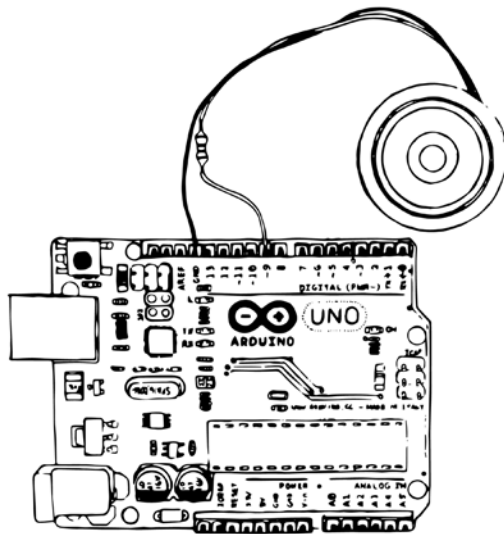
Hold the mic near sound, and the LEDs will light up in time with the music



### ***What to turn in...***

On Friday for the due date, please use Canvas to hand in a photo of your completed board. We'll demo the boards in class on Tuesday.

### 3 Arduino Sound Program - Part I



For this assignment you'll turn in three programs for your Arduino that make sounds. The three programs need to generate compositions/music/noise in the following three ways:

1. Make sounds directly from the Arduino program. You can use arrays to play a specific melody, or use the `random(<min>, <max>);` function to add randomness to your composition.
2. Use the CdS light sensor as input to your composition. This adds an element of environmental sensitivity. You can think about the values you receive from the light sensor as "correlated randomness." That is, they are a way of influencing the composition in a somewhat random way, but correlated to the amount of light falling on the sensor. You can use the sensor values to influence pitches, or durations, or both.
3. A composition of your own choice that uses whatever techniques you like.

All three compositions should be handed in in two ways:

1. Hand in the Arduino code that you used to make each composition. The Arduino code is in your Arduino directory and has a file extension of `.ino`. Your Arduino directory is under Documents/Arduino on a Mac, or My Documents\Arduino on a Windows machine. Each "sketch" you make (Arduino-speak for "program") lives in the Arduino directory inside a directory that is named for you your program. That is, if I made a program called Composition1, the file to turn in would be inside the Arduino/Composition1 folder, and would be named Composition1.ino.
2. Hand in captured sound samples from each composition. You can capture the sound samples using the microphone on your phone, or on your laptop, or even using your amplifier/recorder if you put the inductive pickup right next to the speaker coil of your tiny speaker connected to your Arduino. The sound samples should be at least 15sec each.

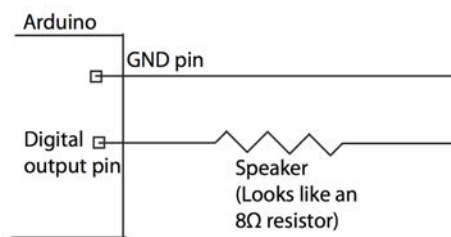
## Preparing your Speaker

Your tiny little speaker is a standard  $8\Omega$  speaker. That means that if you measure the resistance between the red and black wires of the speaker, it will measure  $8\Omega$  of resistance. If you connect the speaker in a circuit, it will look to the rest of the circuit like an  $8\Omega$  resistor.

Actually it's a little more complicated than that because speakers represent a slightly variable resistance depending on what frequency you're sending it, but it's close enough!

You will be connecting one wire of your speaker to a digital output pin on your Arduino. The other wire will be connected to ground (GND) in the Arduino. Your Arduino will be sending audio signals to your speaker by changing the voltage on that digital output pin from 0v to +5v at whatever frequency you want the speaker make.

So, if you think about it, when your Arduino is setting the pin high, there is a path for current to flow from the digital output pin, through the speaker, to ground. We can use Ohm's Law to figure out how much current will flow in that circuit:



Ohm's Law:  $V = IR$

So,  $I = V/R = 5v/8\Omega = 0.625A$

The only problem here is that 0.625A is much too much current for the poor little Arduino to supply! 0.625A is the same as 625mA – remember that 1mA (milli-Amp) is 1/1000 of an amp.

It turns out that the Arduino can supply a maximum of 40mA on each digital pin. That's 0.040A which is a lot less than 0.625A!

To fix this we need to add another resistor in series with the speaker. Resistances in series add together. That is, if we put a  $100\Omega$  resistor in series with an  $8\Omega$  resistor, the total resistance will be  $108\Omega$  in that circuit.

We can use Ohm's Law again to figure out what the right total resistance should be to not exceed the 40mA max that the Arduino can supply.

$V=IR$  So,  $R=V/I=5v/0.040A=125\Omega$

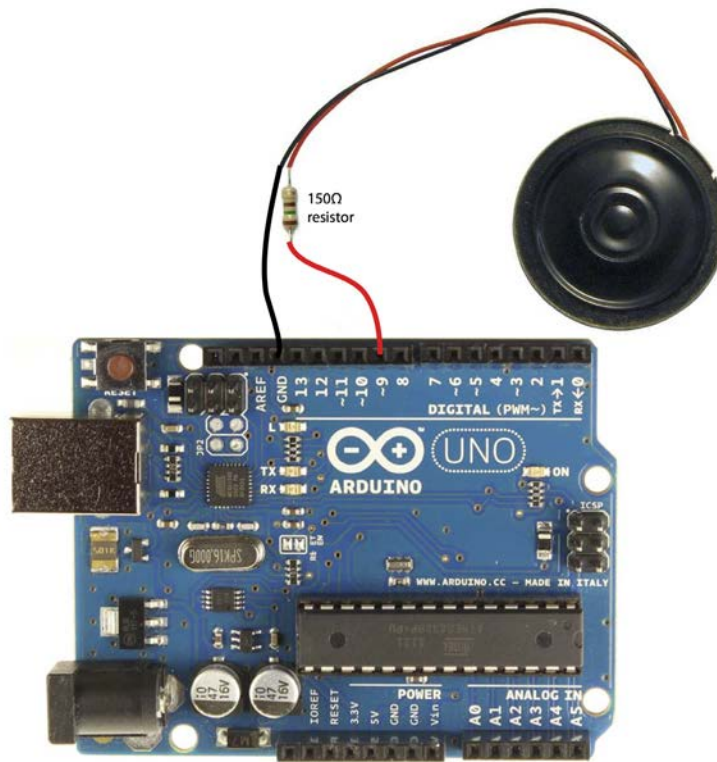
This means that with  $125\Omega$  of total resistance, the Arduino will be asked to supply exactly 0.040A of current.

We'll be safer by using a slightly larger resistor to make sure we don't even get close to the 0.040A max. That's why we're using a  $150\Omega$  resistor in series with your speaker. That will provide a total of  $158\Omega$  resistance in the circuit, and  $I = V/R = 5v/158\Omega = 0.032A$  (32mA). That's nice and safe, and will still be loud enough to hear.

You should take your speaker, and solder the 150Ω resistor to one of the leads. Kirchhoff's current law says that all components in series will see the same current, so it doesn't matter if you solder the resistor to the red or the black wire.

I recommend that you also solder some 20 gauge solid-core wire to each of the speaker wires so that they'll be easier to stick into the holes in the Arduino board. Remember to shrink-wrap your solder connections so that they'll be insulated, and be a little stronger physically.

Here's what your speaker/resistor/Arduino connection should look like:



## Preparing your CdS Light Sensor

Your CdS light sensor is essentially a small resistor that changes its resistance depending on how much light falls on it. The more light that falls on the squiggly line in the face of the sensor, the smaller the resistance. The darker it is, the more resistance there is. This is very nice because you can now use this behavior to sense how much light is falling in the environment in which you've installed the sensor.

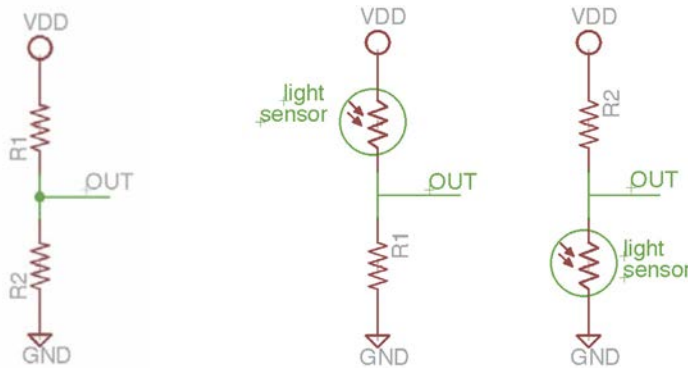
The (slight) problem is that it's a little tricky to measure resistance directly using Arduino. The good news is that it's very easy to sense analog voltages using Arduino. That's what the analog input pins are for. They are connected to circuits called "Analog to Digital Converters" also known as ADC circuits. An ADC will measure the voltage (between 0v and 5v in our case), and return a number (integer) that represents where the voltage falls in that range.

The ADCs on Arduino have 10 binary bits of precision. That means that they can return numbers between 0 and 1023. So, if the voltage on the analog input pin is 0v, the ADC will return 0. If the voltage is 5v it will return 1023. If it's 2.5v it will return a number halfway between – so it will return 511.

Luckily for us, there's a very simple circuit that takes resistances and converts the ratio of resistors to a variable voltage – this is called a Voltage Divider. We saw this in class. It's basically two resistors in series with 5v on one end, and

GND on the other. The voltage is dropped across each of the two resistors in proportion to the size of the resistors. So, the voltage in the middle is related to the ratio of resistances of the two resistors. If both resistors are equal, the voltage in the middle will be exactly half of the power supply voltage (2.5v in this case).

Here's what a general voltage divider looks like (fig xa)



The formula for the voltage at OUT (with respect to GND) is  $V_{out} = V_{dd} (R_2 / (R_1 + R_2))$ . That is, the output is the ratio of  $R_2$  to the total resistance ( $R_1 + R_2$ ) multiplied by the original power supply voltage  $V_{dd}$ .

Now if you make one of the resistors variable, then the output voltage at OUT changes as the resistance of the variable resistor changes. This is excellent, because a changing voltage is exactly that the ADC in the Arduino senses! Here's what the CdS light sensor version of the voltage divider looks like (fig xb).

Note that really doesn't matter whether the variable resistor (the light sensor) is closer to  $V_{dd}$  or closer to GND – because these are resistors in series, they will both see the same current, and both drop voltage across the components. The only functional difference is whether the voltages at OUT range from (for example) 0v to 3v, or from 2v to 5v. The circuit on the left will give voltages up to 5v, but not all the way to 0v. The circuit on the right is the opposite – it will return voltages at OUT as low as 0v, but not all the way to 5v.

Note that we should ask the current question here! With this circuit, how much current will flow from  $V_{DD}$  to GND? Back to Ohm's Law!

$$V = IR \text{ So, } I = V/R$$

In this case we really want as little current to flow as possible. That will burn less power. The trade off is that with tiny currents, the change in voltage will be slower as the light sensor changes. But, that's probably fine. Slow in this case is still fast for humans...

One problem is that we don't know the actual range of resistances in the CdS light sensors! They're all different. So, let's assume that resistance can go all the way to  $0\Omega$ . It won't, but that's the worst case. In that case, all the resistance in the circuit is in the other (fixed) resistor.

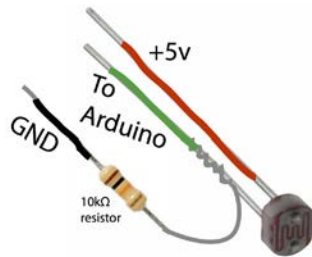
In these cases, it's very common to use a  $10,000\Omega$  (10k $\Omega$ ) resistor. That's large enough to really limit the current, and not so large that things get noticeably slow.

Let's see how much current that really is:  $I = V/R = 5v/10,000\Omega = 0.0005A = 0.5mA$ . Half a milli-Amp is a really nice small number! That will work well.

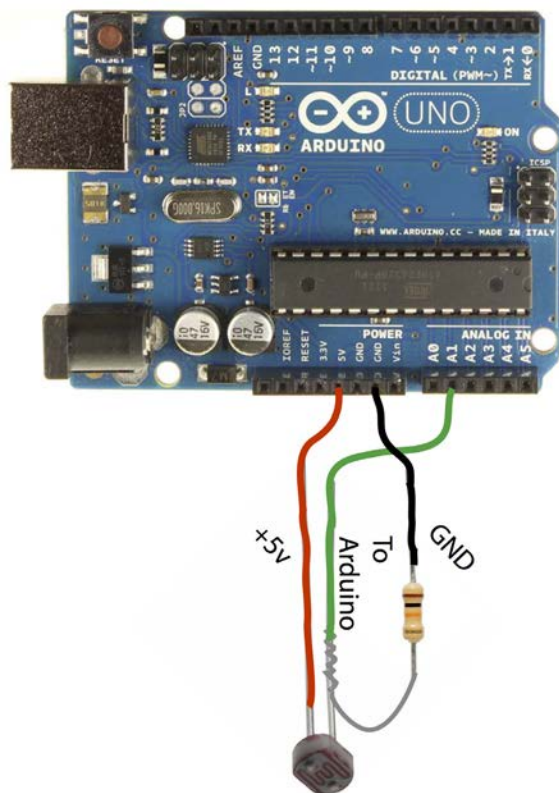
So, you need to do a little more soldering. You need to solder your CdS light sensor, and a 10k $\Omega$  resistor to look like the circuit on the previous page. I recommend also soldering some 20 gauge solid-core wire on the leads so that they're easy to plug into the Arduino – they are also insulated so that they won't short out. Remember the shrink wrap too.



The circuit looks like this:



Here's what your CdS sensor looks like when you connect it to your Arduino:



## Program Examples

On the Canvas page for this class I've put some example programs to get you started. To use these programs make folders in your Arduino directory, and then put the <name>.ino and <name>.h files inside those folders. There are details in the following section "A Brief Introduction to Programming."

The example programs are:

- SimpleTone: This shows extremely simple tone making. It uses the "tone" function to make a series of five tones at particular frequencies. The frequencies are hard-coded with numbers, in Hertz.

### SimpleTone.ino

```
/* VERY simple tone program */
int speakerPin = 9; // attach the speaker to pin 9

void setup(){
  pinMode(speakerPin, OUTPUT); // Make speakerPin an output
}

void loop(){
  tone(speakerPin, 440); // tone fires up at 440Hz
  delay(1000);           // play it for 1 sec
  tone(speakerPin, 494);
  delay(1000);
  tone(speakerPin, 131);
  delay(500);
  tone(speakerPin, 554);
  delay(2000);
  tone(speakerPin, 143);
  delay(1000);
}
```

- SimpleTone1: This version adds two things:
  - a. It uses the "pitches.h" file to "name" the notes. That is, it gives names to each of the pitch frequencies so that you can call standard pitches by their name instead of knowing their frequencies
  - b. It shows how to make silence between notes using the noTone() function to turn off the signal to the pin.

### SimpleTone1.ino

```
/* VERY simple tone program */
#include "pitches.h"
int speakerPin = 9; // attach the speaker to pin 9
```



```

void setup(){
  pinMode(speakerPin, OUTPUT); // Make speakerPin an output
}

void loop(){
  tone(speakerPin, NOTE_A4); // tone fires up an A4
  delay(1000);                // play it for 1 sec
  noTone(speakerPin);         // stop the tone
  delay(300);                 // “play” some silence
  tone(speakerPin, NOTE_B4); // play another tone
  delay(1000);
  tone(speakerPin, NOTE_C3);
  delay(500);
  tone(speakerPin, NOTE_CS5);
  delay(2000);
  tone(speakerPin, NOTE_D3);
  delay(1000);
}

```

- SimpleTone2: This uses a slightly different way of deciding how long a note plays. This is not that different from SimpleTone1, and you can ignore it if you like. I find this version a little counter-intuitive, but am including it for completeness.

### **SimpleTone2.ino**

```

/* VERY simple tone program */
#include "pitches.h"
int speakerPin = 9; // attch the speaker to pin 9

void setup(){
  pinMode(speakerPin, OUTPUT); // Make speakerPin an output
}

void loop(){
  tone(speakerPin, NOTE_A4, 1000); // tone fires an A4 for 1sec
  delay(1500);                      // delay for 1.5sec for...
  tone(speakerPin, NOTE_B4, 1000); // some space
  delay(1500);
  tone(speakerPin, NOTE_C3, 500);
  delay(700);
  tone(speakerPin, NOTE_CS5, 1500);
  delay(2000);
  tone(speakerPin, NOTE_D3, 1000);
  delay(1300);
}

```

- **toneMelody:** This is not on the Canvas page because it's in the examples that come with the Arduino programming interface. It's in the Examples-Digital- toneMelody menu in your Arduino program. This program also uses pitches.h and shows how to make arrays that hold note values and duration values.

### **toneMelody.ino**

```
/* Melody Plays a melody
circuit:
* 8-ohm speaker on digital pin 8

created 21 Jan 2010
modified 30 Aug 2011
by Tom Igoe

This example code is in the public domain.
http://www.arduino.cc/en/Tutorial/Tone

*/
#include "pitches.h"

// 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(8, 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(8);
  }
}

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

- RandomMelody: This is a version of the toneMelody program, but one that uses the random() function to choose notes and durations randomly from the arrays to make a random melody instead of the “shave and a haircut, two bits” melody from toneMelody.

### RandomMelody.ino

```
// use the pitches.h file to select some notes to choose from
// then select a random duration
// and play random sequences of these notes.
#include "pitches.h"
int speakerPin = 9; //
int randomNote; // variable to hold which note you're playing
int randomDuration; // variable to hold random duration
int noteDuration; // The duration scaled by 1sec

// notes to choose from (array goes from 0 to 7)
// Note that one of the choices is 0 meaning "play nothing"
int notes[] = {
    NOTE_A4, NOTE_B3, NOTE_C3, NOTE_D3, NOTE_E3, NOTE_F3,
    NOTE_G3, 0};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
// Note that by putting more 8's and 4's, those durations will
// be more common when chosen at random from the array
// This array goes from 0 to 9
int durations[] = {
    1, 2, 4, 4, 4, 8, 8, 8, 16, 16 };

void setup() {
    pinMode(speakerPin, OUTPUT); // make sure it's an output pin
}

void loop() {
    // Choose a random note
    randomNote = notes[random(0, 8)]; // remember (min, max-1)
    // and a random duration
    randomDuration = durations[random(0,10)];

    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    noteDuration = 1000/randomDuration;

    tone(speakerPin, randomNote); // now play a random note
    delay(noteDuration);           // and delay by a random duration
}
```

- Calibration: This program shows how to use the Serial library to output the values that you're getting back from the light sensor. This is a way to calibrate the highest and lowest value you can expect from the current environment in which you've installed your light sensor.

#### **Calibration.ino**

```
/*
 * This program demonstrates how to calibrate a resistive sensor by
 * printing the values you get back from the sensor to the serial
 * monitor. You can eyeball the values and get a good feel for the
 * range of values you can expect from the sensor.
 */

int sensorPin = A0; // analog input pin for the potentiometer
int sensorValue = 0; // variable to store value from the sensor

void setup() {
  Serial.begin(9600); // Init serial communication at 9600 baud
}

void loop() {
  sensorValue = analogRead(sensorPin); // read the value from the sensor:
  Serial.print("Sensor value is: "); // print a message (no newline yet)
  Serial.println(sensorValue); // print the value you got (with new line
  delay(100); // wait so you don't print too much!
}

// VERY useful for getting a feel for the range of values coming in
// Remember to open the Serial Monitor to see the values
```

- tonePitchFollower2: This will take the analog sensor values from your light sensor and make a Theremin-like sound on your speaker. That is, the pitch will follow the amount of light on your light sensor.

#### **tonePitchFollower2.ino**

```
/* Pitch follower

Plays a pitch that changes based on a changing analog input
circuit:
 * 8-ohm speaker on digital pin 8
 * photoresistor on analog 0 to 5V
 * 4.7K resistor on analog 0 to ground

created 21 Jan 2010
modified 31 May 2012
by Tom Igoe, with suggestion from Michael Flynn
```

This example code is in the public domain.

```
http://arduino.cc/en/Tutorial/Tone2  */

void setup() {
  // initialize serial communications (for debugging only):
  Serial.begin(9600);
}

void loop() {
  // read the sensor:
  int sensorReading = analogRead(A0);
  // print the sensor reading so you know its range
  Serial.println(sensorReading);
  // map the analog input range
  // (in this case, 400 - 1000 from the photoresistor)
  // to the output pitch range
  // (in this case 120 - 1500Hz)
  // change the minimum and maximum input numbers below
  // depending on the range your sensor's giving:
  int thisPitch = map(sensorReading, 400, 1000, 120, 1500);

  tone(9, thisPitch); // play the pitch:
  delay(10);          // play for 10ms, then resample
}
```

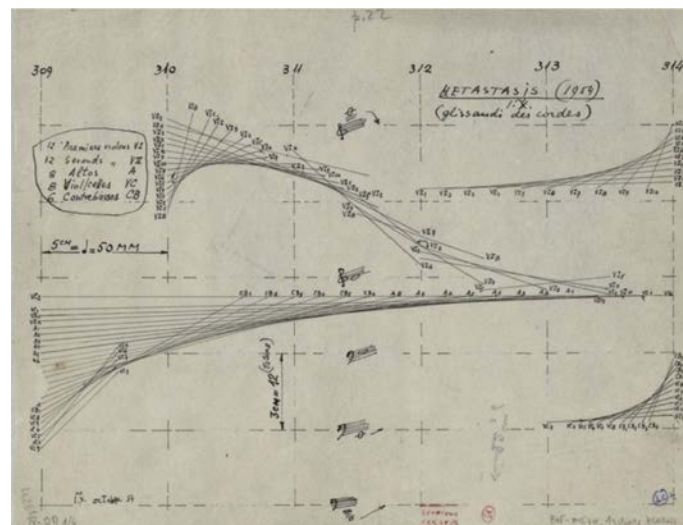
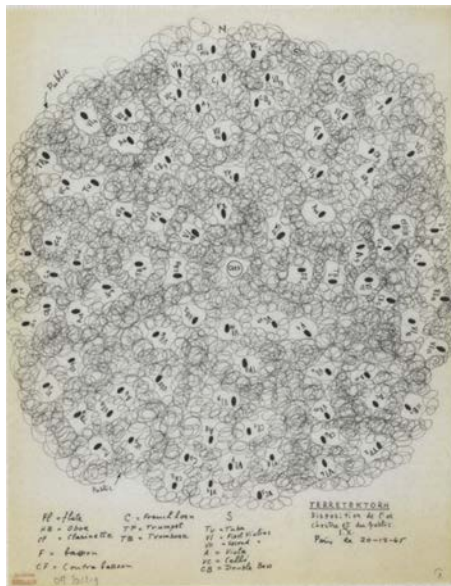
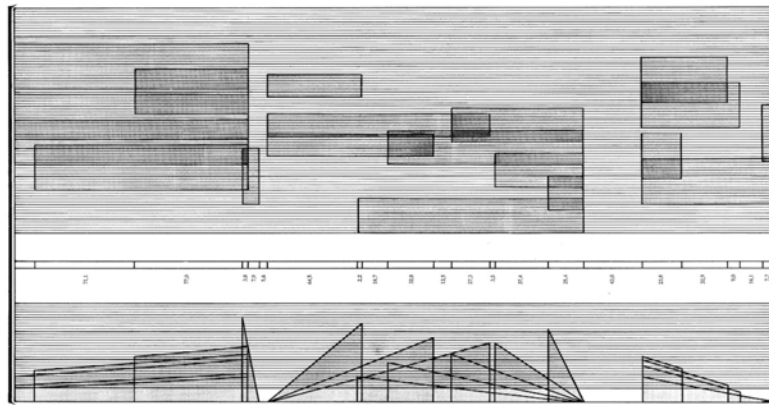
*Good luck – have fun – try things – and let us know if you have problems or questions!*

### ***What to turn in...(repeated from pg. 27)***

All three compositions should be handed in in two ways:

1. Hand in the Arduino code that you used to make each composition. The Arduino code is in your Arduino directory and has a file extension of .ino. Your Arduino directory is under Documents/Arduino on a Mac, or My Documents\Arduino on a Windows machine. Each “sketch” you make (Arduino-speak for “program”) lives in the Arduino directory inside a directory that is named for you your program. That is, if I made a program called Composition1, the file to turn in would be inside the Arduino/Composition1 folder, and would be named Composition1.ino.
2. Hand in captured sound samples from each composition. You can capture the sound samples using the microphone on your phone, or on your laptop, or even using your amplifier/recorder if you put the inductive pickup right next to the speaker coil of your tiny speaker connected to your Arduino. The sound samples should be at least 15sec each.

# 4 Arduino Sound Program - Part II



Experimental scores for computer music. Top row: Stockhausen; bottom row: Xenakis.

At this point you've seen a variety of abstract scores for computer generated music - a few of which are shown in the above figure. In the second part of this assignment, compose another short piece (music/sounds/noise), and do the following:

1. Draw an abstract score for your piece, in whatever notation you'd like.
2. Write the code to implement the score, and include a description about how the score and the program are connected.
3. Make a 15 second recording of your piece using the same methods described in part I.
4. Import your sound recordings into audacity and play with some filters. What difference do you see in the shape of the sound waves? How might this correspond to the noises you're hearing?

***What to turn in...***

Your composition should be handed in in several ways:

1. Scan and upload an image of your score.
2. Hand in your arduino code, as you did in the previous assignment.
3. Hand in a description about how your score and program are connected. This can be written in your journal and scanned/uploaded to canvas, or written in a separate document.
4. Hand in captured sound samples from your composition, once again, as you did in Part 1 of this assignment.
5. In your journal, jot down any additional observations you make while experimenting with your recording in Audacity.

# 5 Hardware Hacking

For this assignment you should acquire a toy that makes music or noise or speech. You can acquire this toy from a thrift store (such as Deseret Industries where toys of this sort are typically \$1 or \$2), or steal it from your younger siblings. Don't steal anything that they'll miss though - the toy will never be the same after you're finished with it. Take the toy apart and hack it to make new sounds that it was never intended to make. We have lots of supplies to help you: potentiometers, wires, resistors, diodes, etc. We also have a bunch of cigar boxes you can use to repackage your new noise instrument.

**Reading:** This project is based on material in Part III (Chapters 12-17) in our textbook.

There are on-line versions of the videos from the book that show examples of two hacks. We'll go over these in class, but you might want to watch them on your own time too:

Laying of Hands on a battery powered radio:

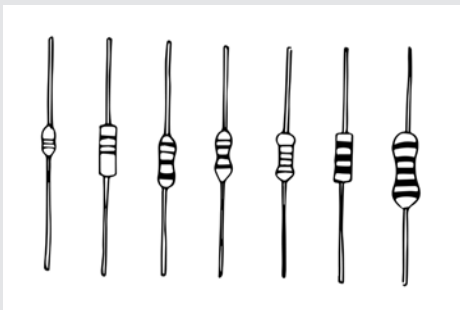
<http://www.nicolascollins.com/hackingtutorial09.htm> (Links to an external site.)

Tickling the clock on a noise-making toy:

<http://www.nicolascollins.com/hackingtutorial10.htm>

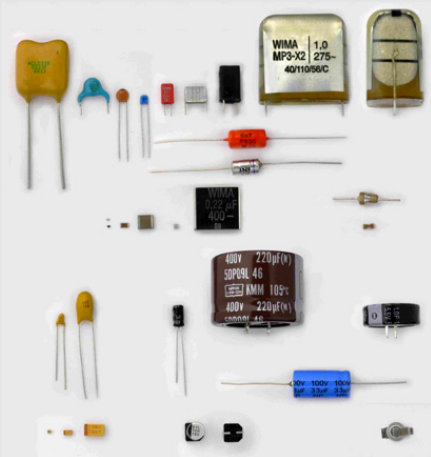
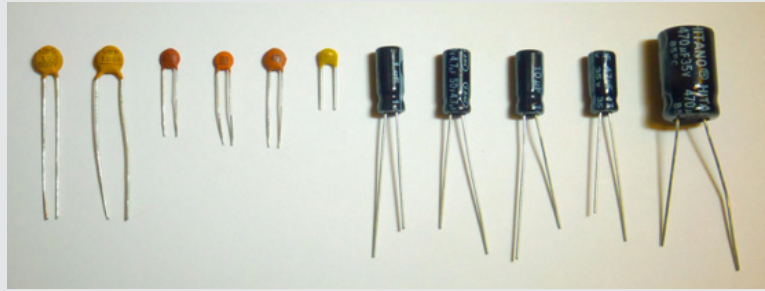
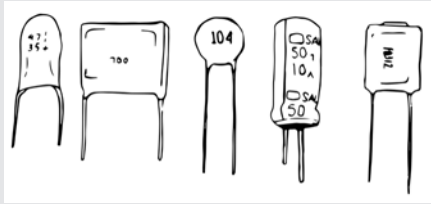
The following section provides some details on the various components you might encounter in your circuit bending.

## RESISTORS

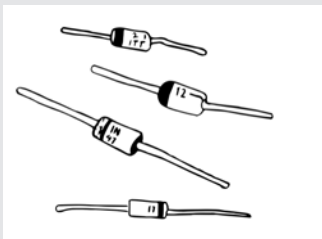




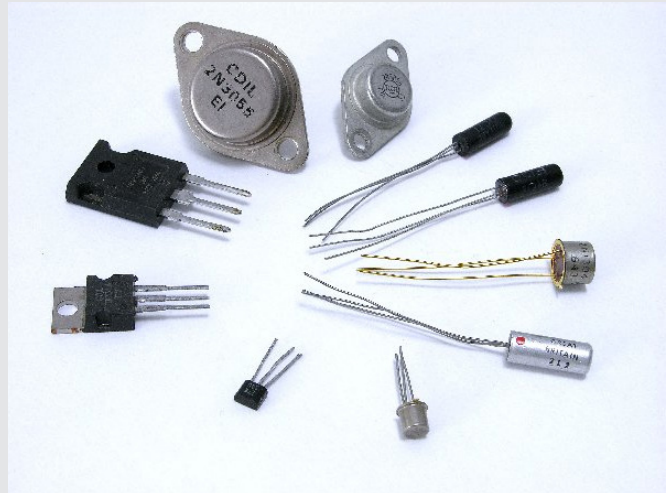
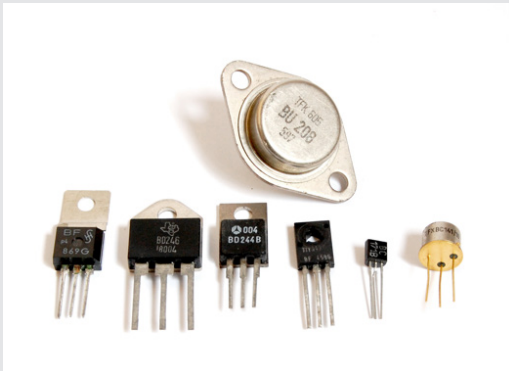
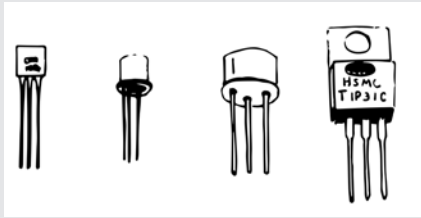
## CAPACITORS



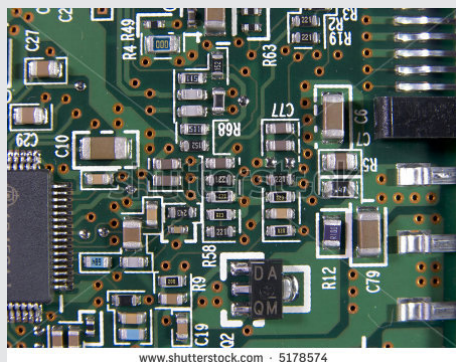
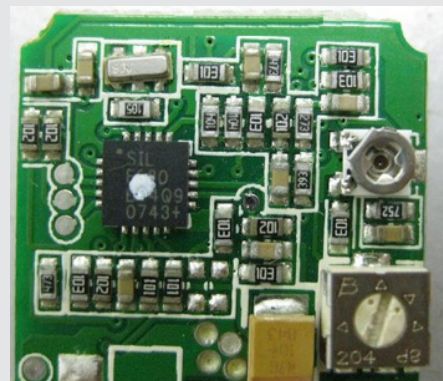
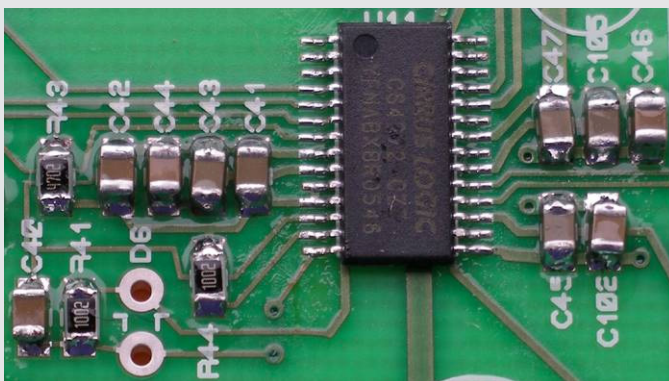
## DIODES



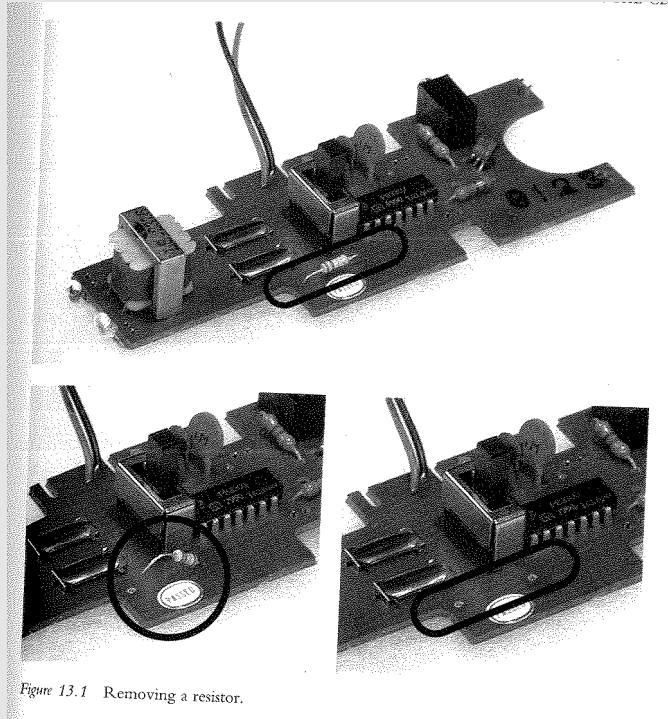
## TRANSISTORS



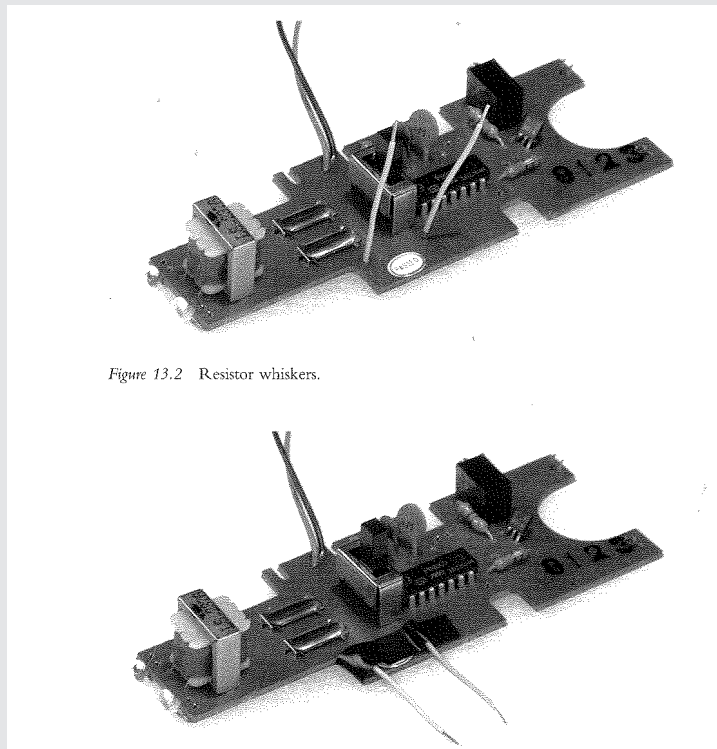
## SURFACE MOUNT COMPONENTS



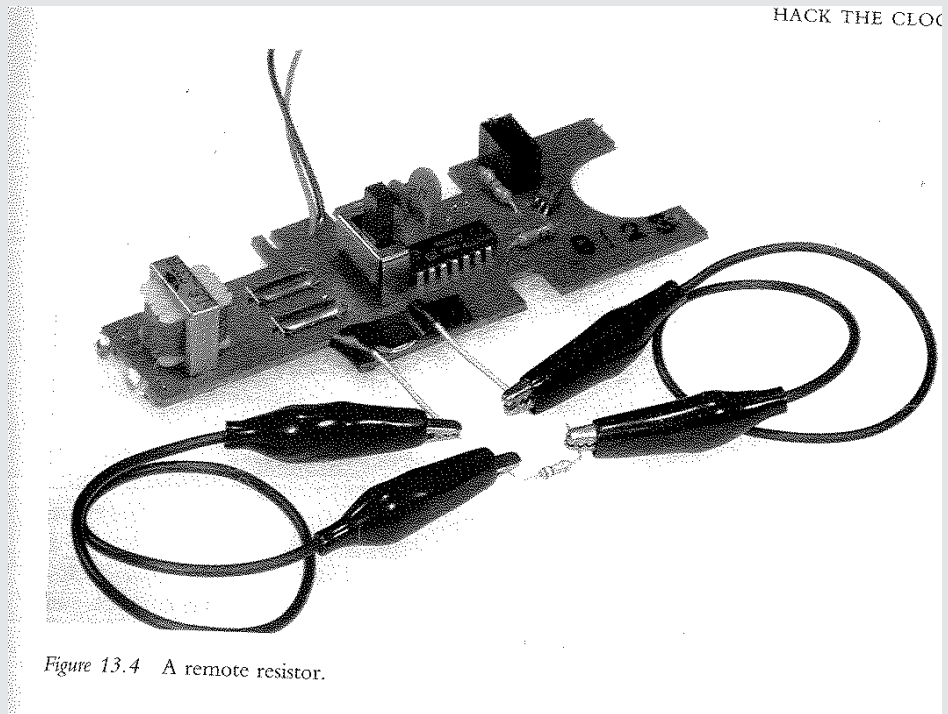
## CLOCK RESISTORS



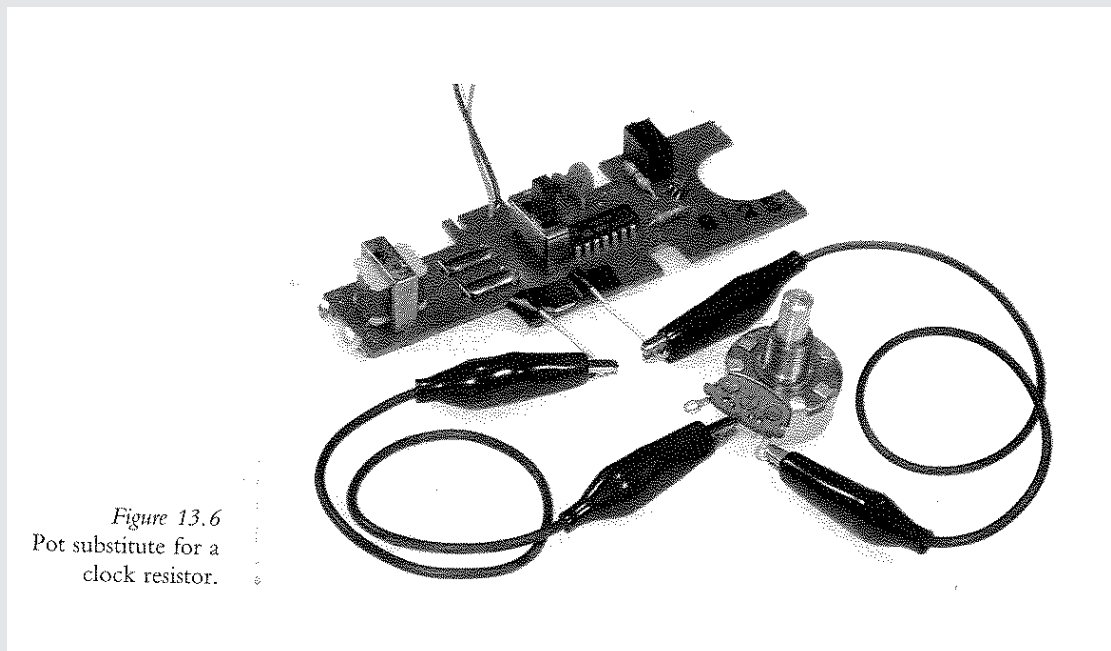
## RESISTOR WHISKERS



## ALLIGATOR CLIPS

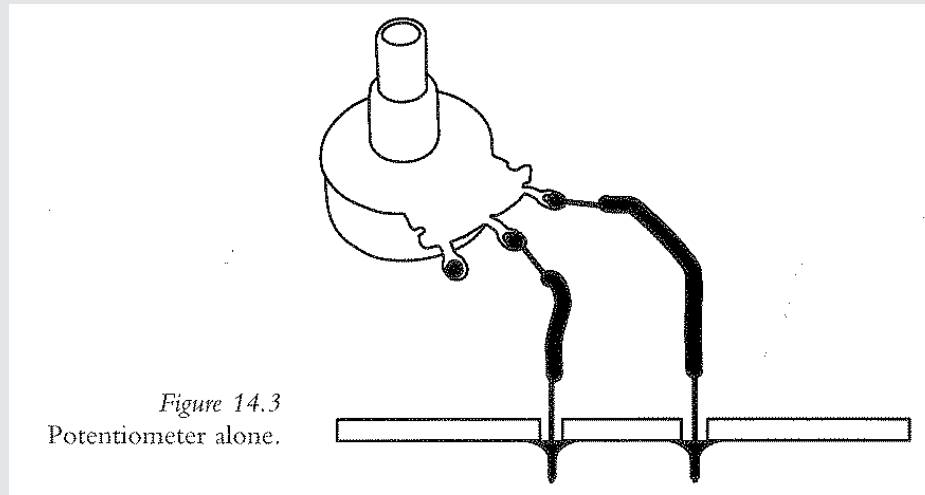


## POT CONNECTION





## POT CONNECTION



### ***What to turn in...***

1. Write up instructions on how to play your toy. You can write these instruction up in your journal and scan/upload them to Canvas.
2. Hand in captured sound samples from your toy. Upload your sound recordings to SoundCloud and turn in links on Canvas.
3. Plan to demo your bent toy in class on the due date.

# A Breif Introduction to Programming

## Disclaimer

Much of this material is mine. Some is taken from various places on the web:

- todbot.com – Bionic Arduino and Spooky Arduino class notes from Tod E.Kurt
- ladyada.net – Arduino tutorials by Limor Fried

## To start, what is a program?

- Essentially just a list of actions to take
- Each line of the program is step to take
- The program just walks through the steps one at a time
- Maybe looping too

*It's like a recipe!*

Take meatloaf, for example:

### Meatloaf Recipe Ingredients:

1 package Lipton Onion Soup Mix  
2 pounds lean ground beef  
1 large egg  
2/3 cup milk  
3 Tablespoons catsup  
3 Tablespoons brown sugar  
1 Tablespoon yellow mustard

### Directions:

1. Preheat the oven to 350 degrees F.
2. Mix the onion soup mix, ground beef, egg and milk together.
3. Form the combination into a loaf shape in a 13 X 9 X 2 loaf pan.
4. Combine the rest of the ingredients and spoon onto the top of the meatloaf.
5. Bake uncovered, for about an hour.
6. When done, take the meatloaf out of the pan and place on a serving plate. Let stand for 10 minutes before slicing.

Or shampoo directions:

1. Lather
2. Rinse
3. Repeat

*But when do you stop?*

### *What about the following?*

1. Lather
2. Rinse
3. If this is the first lather, then Repeat else stop and towel off



We can write this as the following:

```
Repeat twice {  
  Lather  
  Rinse  
}
```

Or as this:

```
For (count=1; count<3;  
count=count+1) {  
  Lather  
  Rinse  
}  
  
//The above would executes as follows  
count=1  
lather  
rinse  
count=2  
lather  
rinse  
count=3  
continue to next instruction...
```

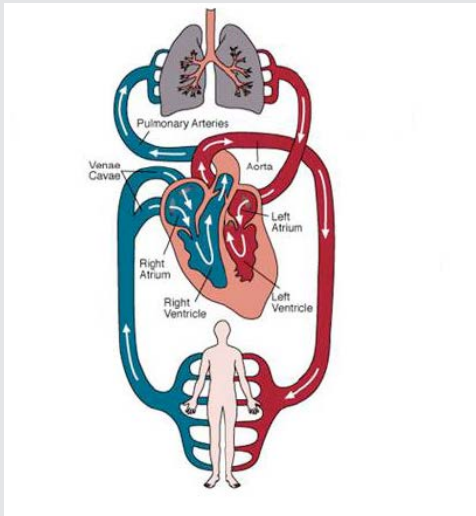
Ok, lets move onto something a little more interesting.

```
//make a flashlight  
1. Turn light on  
2. Wait for 1 second  
3. Turn light off  
4. Wait for one second  
5. repeat
```

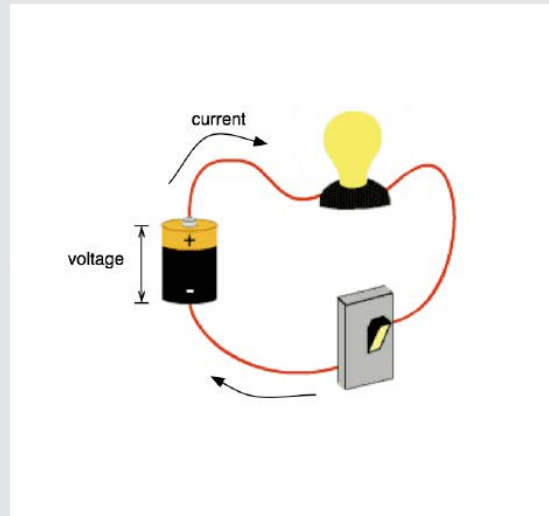
We'll come back to this...Let's first talk about lights!

First, electricity!!

## Circuits



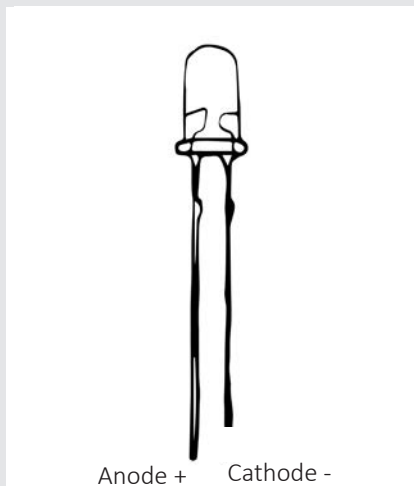
Heart pumps, blood flows



Voltage pushes, current flows

## Making A Flightlight - components

### LED's



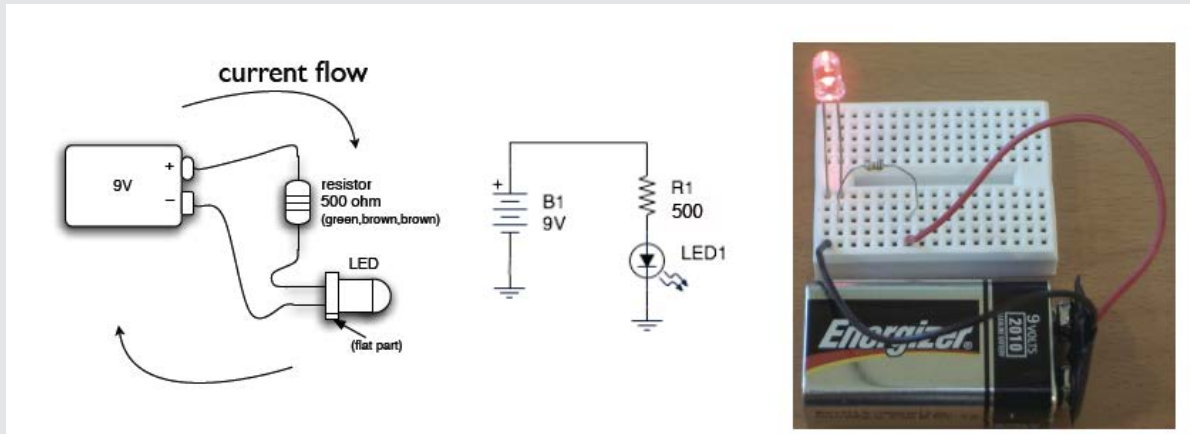
Current flows from Anode to Cathode  
Lights up when current flows

### Resistors



Polarity doesn't matter on resistors.

## Making A Flightlight - wiring it up



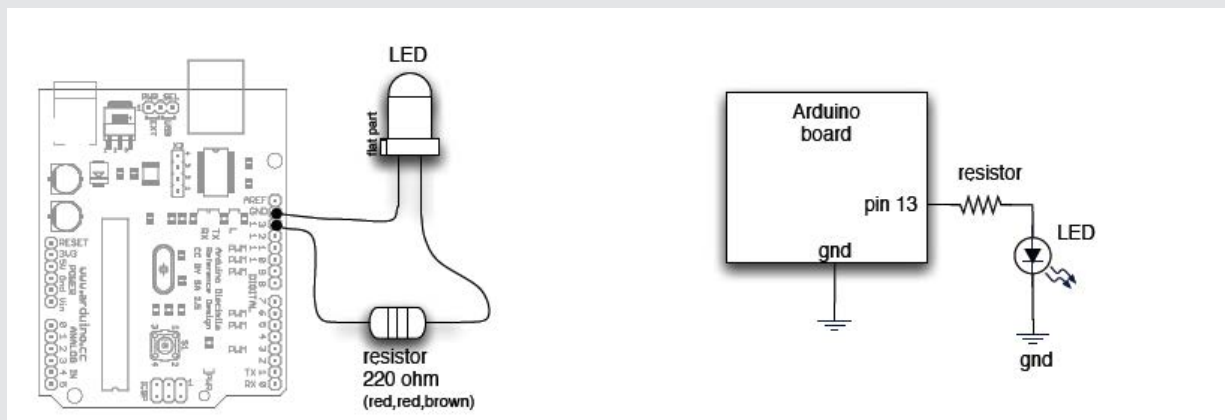
wiring diagram

schematic

wiring it up

Electricity flow is in a loop. Can stop flow by breaking the loop.

## Making A Flightlight - wiring it up

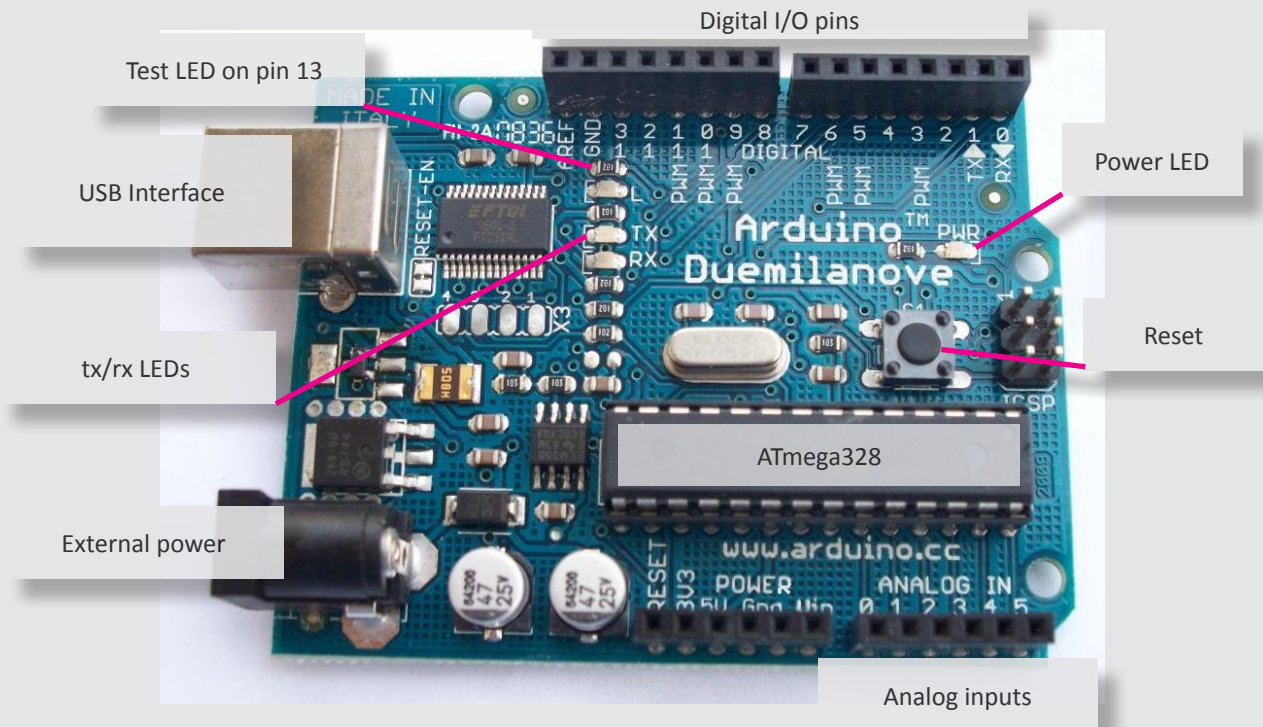


wiring diagram

schematic

Arduino Diecimila board has this circuit built-in. To turn on LED use `digitalWrite(13, HIGH)`

## Arduino



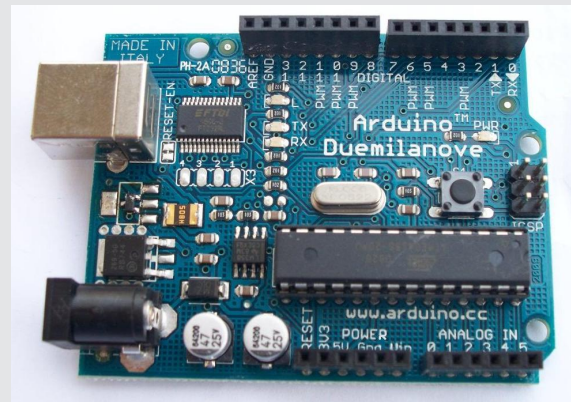
## Arduino Programming



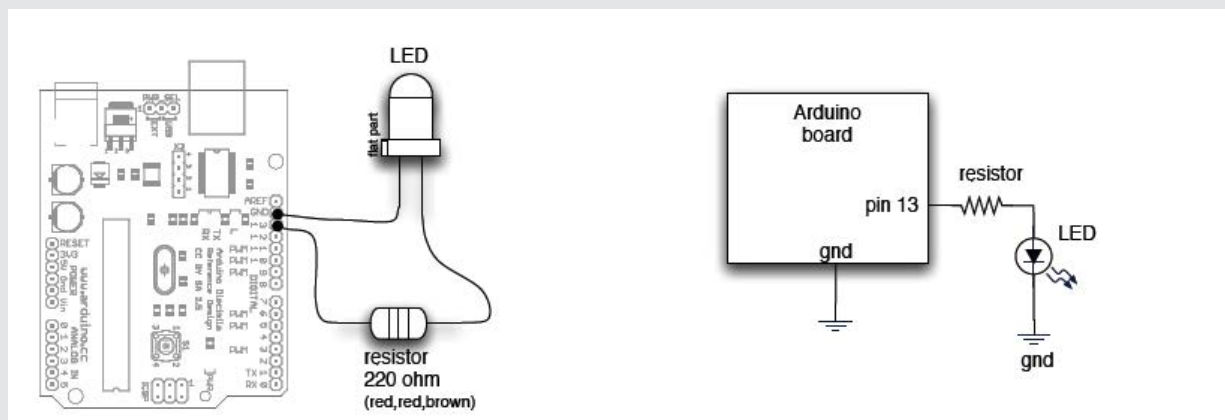
## Digital Pins

Each of the digital pins can be set to one of two values:

- High and Low (+5v and 0v)
- `digitalWrite(<pin-number>, <value>);`
- `digitalWrite(13, HIGH);`
- `digitalWrite(13, LOW);`



## Wiring it up



wiring diagram

schematic

Arduino Diecimila board has this circuit built-in. To turn on LED use `digitalWrite(13, HIGH)`

## Back to our flashlight

```
//make a flashlight
1. Turn light on
2. Wait for 1 second
3. Turn light off
4. Wait for one second
5. repeat
```

We can rewrite this in terms of our “digitalWrite” function:

<pre>//make a flashlight 1. digitalWrite(13,HIGH) 2. delay(1sec) 3. digitalWrite(13,LOW) 4. delay(1sec) 5. repeat</pre>	rewritten as a loop:	<pre>//make a flashlight loop() {   digitalWrite(13,HIGH);   delay(1000); //in milliseconds   digitalWrite(13,LOW);   delay(1000); }</pre>
---	----------------------	--

*It's very common to write things in “pseudo-code” (above) before writing the real program!*

## Let's look at what each statement does...

<pre>//make a flashlight void setup(){   pinMode(13,OUTPUT); }</pre>	<pre>//do once at first //select pin 13 as an output</pre>
<pre>loop() {   digitalWrite(13,HIGH);   delay(1000); //in milliseconds   digitalWrite(13,LOW);   delay(1000); }</pre>	<pre>//loop forever //set pin 13 to HIGH //delay 1000ms (1sec) //set pin 13 to LOW //delay 1000ms (1sec) //return to loop</pre>

The setup function executes once at the beginning of the function.



## Writing an Arduino program - required functions

```
/* define global variables here */
void setup() {                               //run once, when the program starts
<initialization statement>;                 //typically pin definitions
...                                         // and other init stuff
<initialization statement>;
}

void loop() {                                // run over and over again
/* define local variables here */
<main loop statement>;                     // the guts of your program
...                                         // which could include calls
<main loop statement>;                     // to other functions...
}
```

“void” means that those functions do not return any values

## Variables

- ❑ Variables are like mailboxes – you can store a value in them and retrieve it later
- ❑ They have a “type”
  - tells you what values can be stored in them

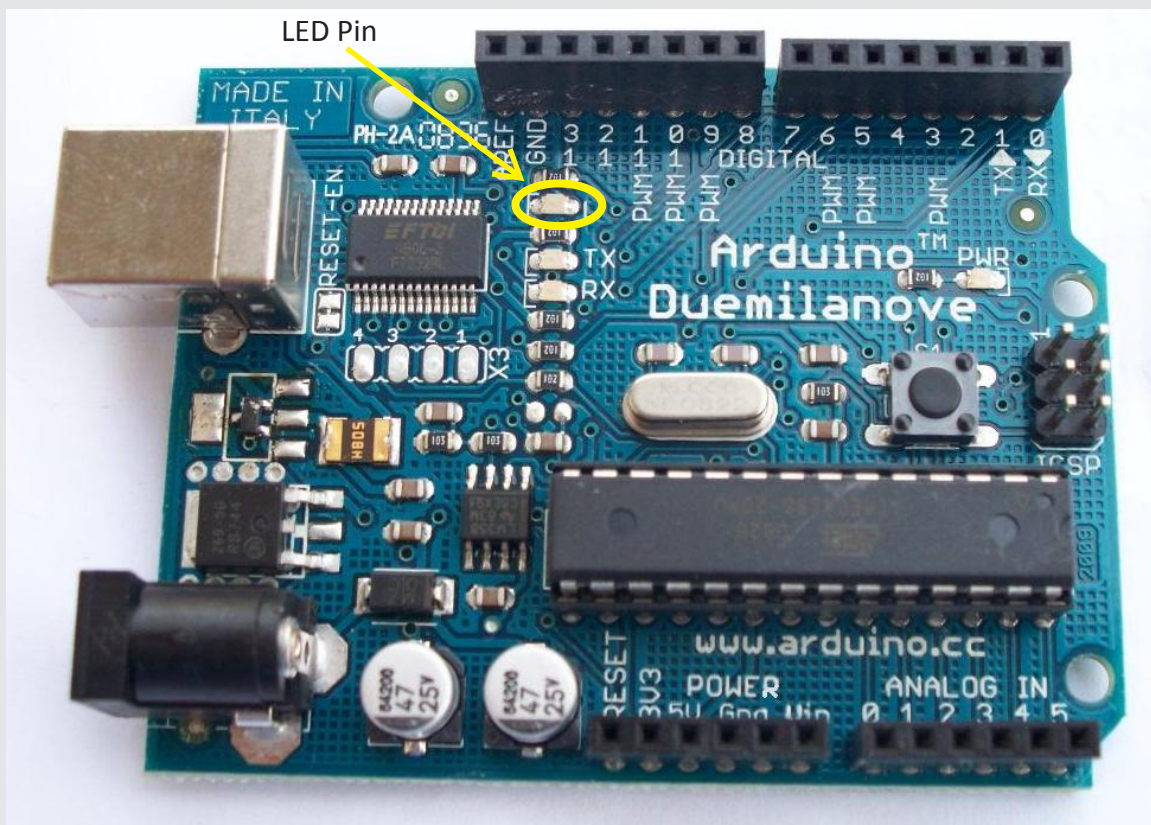
```
// define a variable named “LEDpin”
// start it out with the value 13
int LEDpin = 13;
//you can now use LEDpin in your program
// Wherever you use it, the program will look inside
// and use the 13
```

- ❑ Variable names must start with a letter or underscore
- ❑ variable names are case sensitive!
  - Foo and foo are different variables!
- ❑ After the letter or underscore you can use numbers too
- ❑ Questions: Are these valid names?
  - Abc
  - 1st\_variable
  - \_123\_
  - pinName
  - another name
  - a23-d
  - aNiceVariableName

## Example program

Let's take a look at one of the example programs included in the Arduino software.

```
/*  
 *Blink  
 * The basic Arduino example. Turns on an LED on for one second,  
 * then off for one second, and so on... We use pin 13 because,  
 * depending on your Arduino board, it has either a built-in LED  
 * or a built-in resistor so that you need only an LED.  
 */  
  
int ledPin = 13;           // LED connected to digital pin 13  
  
void setup()  
{  
  // run once, when the sketch starts  
  pinMode(ledPin, OUTPUT); // sets the digital pin as output  
}  
  
void loop()  
{  
  // run over and over again  
  digitalWrite(ledPin, HIGH); // sets the LED on  
  delay(1000);                // wait for a second  
  digitalWrite(ledPin, LOW);  // sets the LED off  
  delay(1000);                // wait for a second  
}
```



## Blink Modifications

- A few things to try with the Blink program:
  - Change so that blink is on for 500msec and off for 100msec  
What happens?
  - Change so that blink is on for 50msec and off for 50msec  
What happens?
  - Change so that blink is on for 10ms and off for 10ms  
What happens?

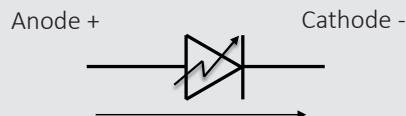
## So...we just made an LED blink...Big Deal?

- Most actuators are switched on and off with a digital output
  - The `digitalWrite(pin,value);` function is the software command that lets you control almost anything
- LEDs are easy!
  - Motors, servos, etc. are a little trickier, but not much
  - More on that later...
- Arduino has 14 digital pins (inputs or outputs)
  - can easily add more with external helper chips
  - More on that later...

## More blink modifications

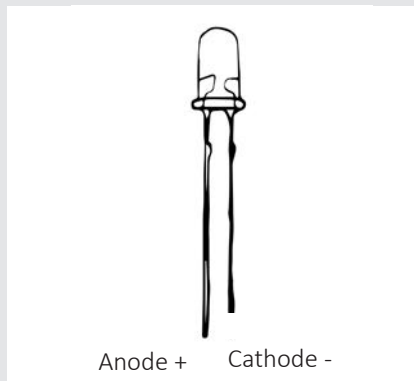
- Change to use an external LED rather than the one on the board
  - Connect to any digital pin
  - LED is on if current flows from Anode to Cathode
  - LED is on if the digital pin is HIGH, off if LOW
  - How much current do you use?
    - not more than 20mA
  - How do you make sure you don't use too much?
    - use a resistor

***Pay attention to current! Use a current-limiting resistor!***



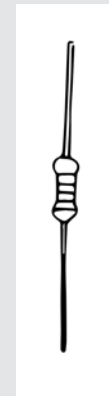
## Making A Flightlight - LEDs and Resistors

### LED's

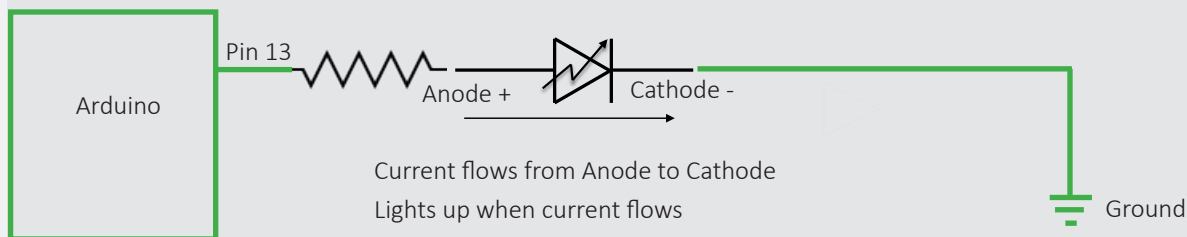


On LED's polarity matters. Shorter side is negative, and goes to ground.

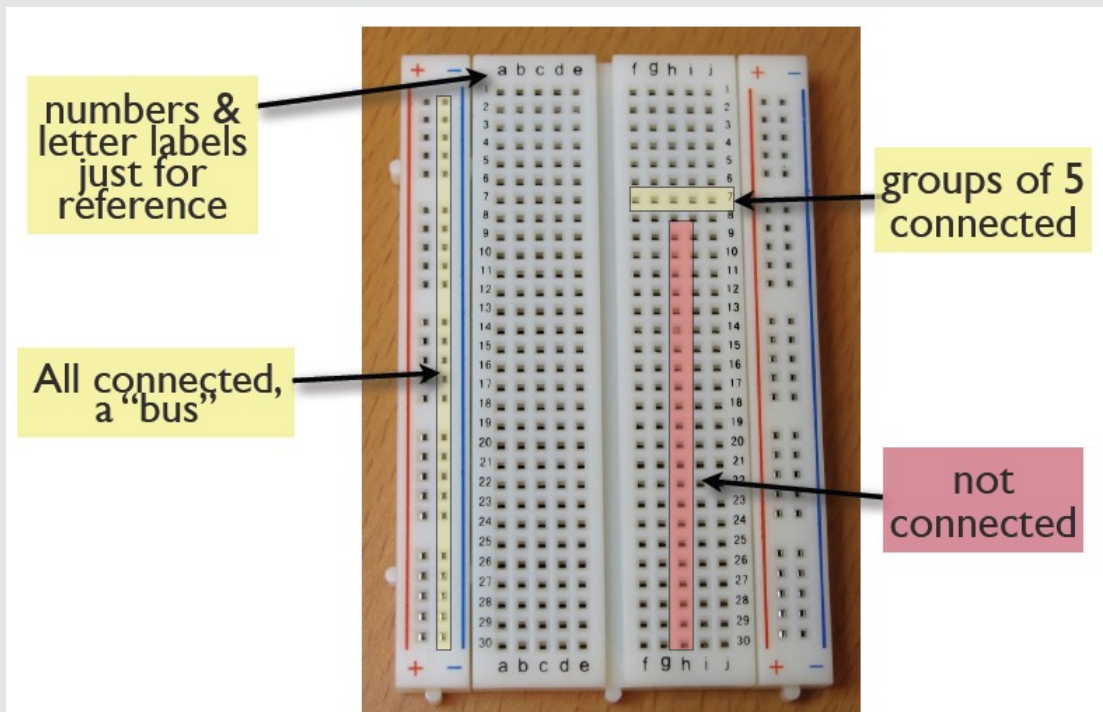
### Resistors



Polarity doesn't matter on resistors.

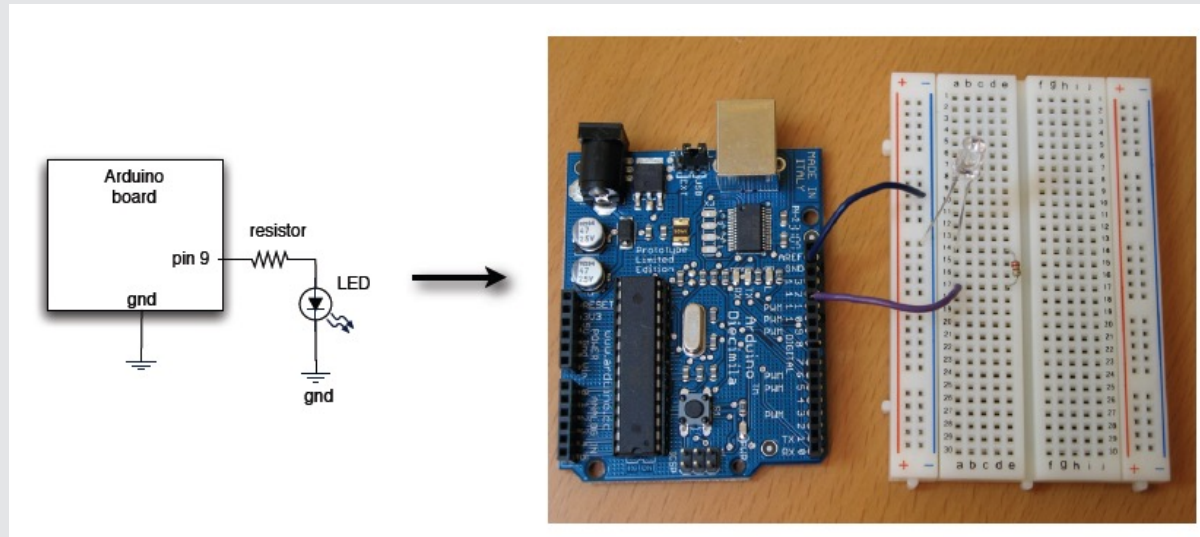


## Proto boards

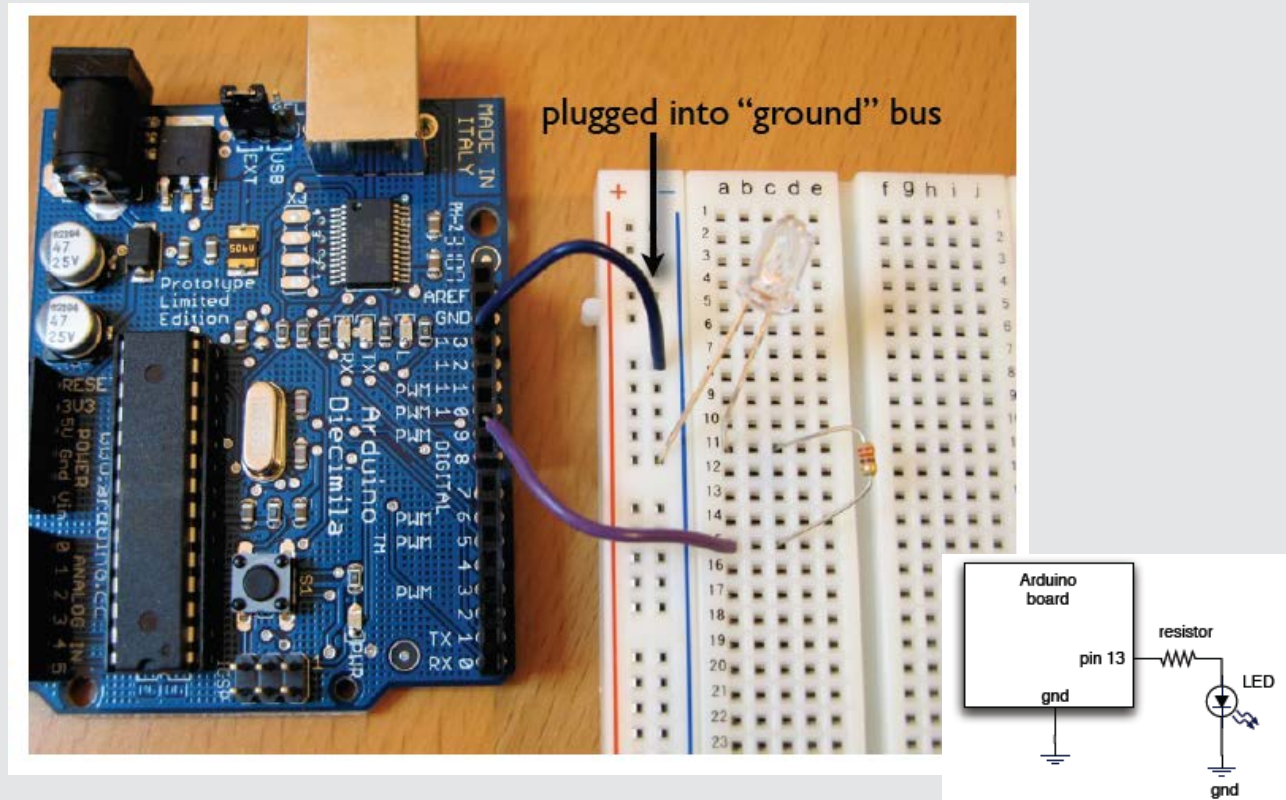


A.K.A. Solderless bread board

Wire it up!

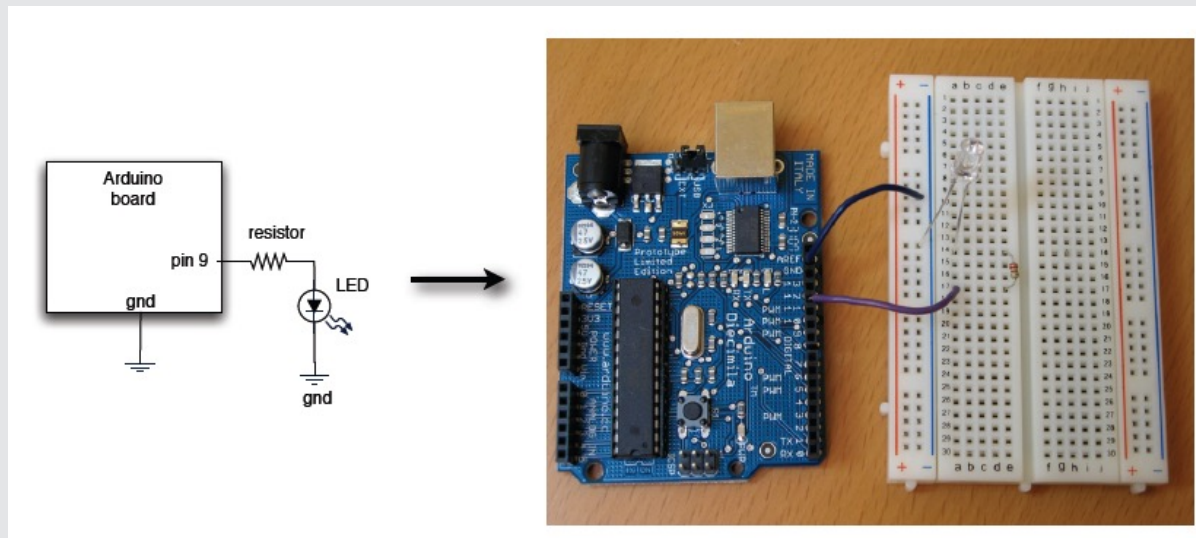


A slightly closer look

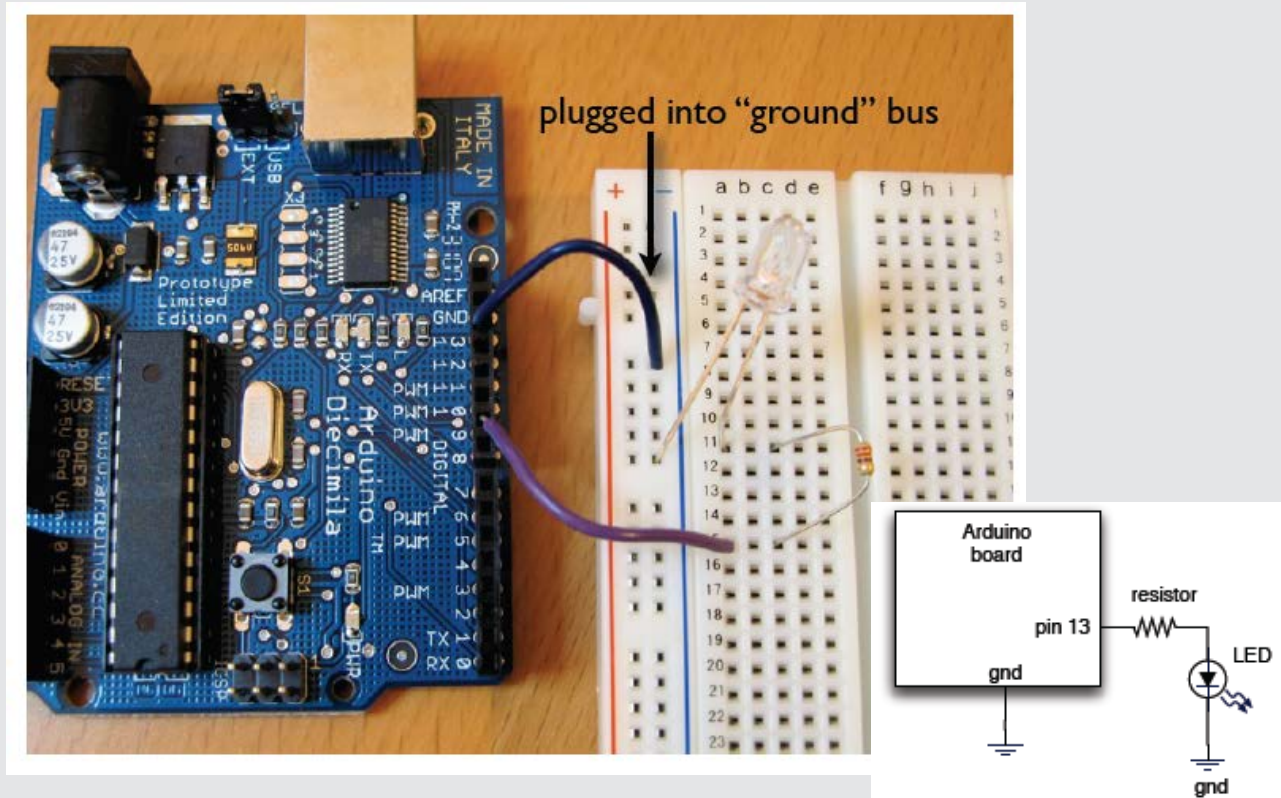




Wire it up!



A slightly closer look





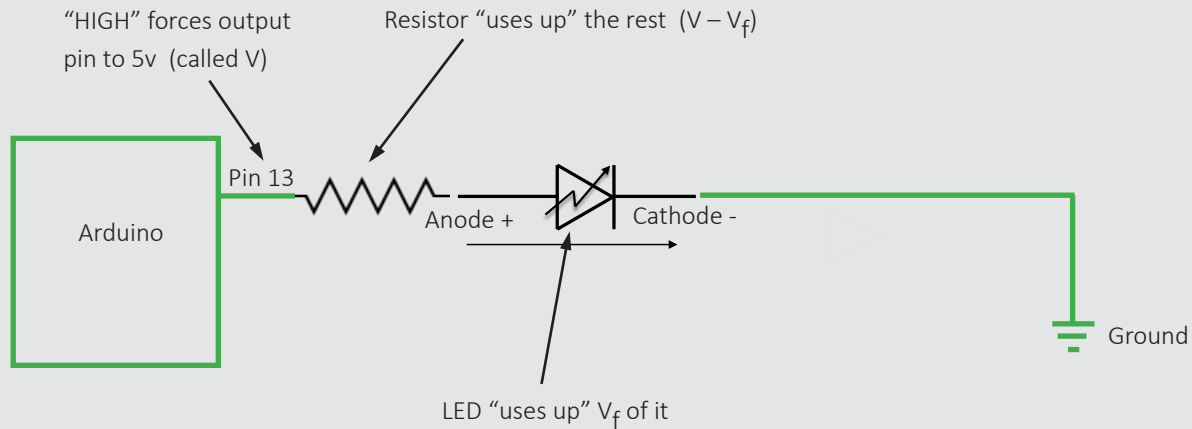
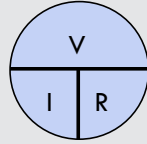
## Current Limiting Resistor

### Ohm's Law

$$- V = IR \quad I = V/R \quad R = V/I$$

### Every LED has a $V_f$ "Forward Voltage"

- How much voltage is dropped (used up) passing through the LED



## Current Limiting Resistor

### Ohm's Law

$$- V = IR \quad I = V/R \quad R = V/I$$

### Every LED has a $V_f$ "Forward Voltage"

- How much voltage is dropped (used up) passing through the LED

### $R = (V - V_f) / I$

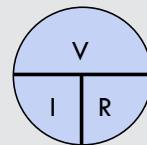
- Example – If  $V_f$  is 1.9v (red LED), and  $V = 5v$ , and you want 15mA of current (0.015A)

$$- R = (5 - 1.9)/0.015 = 3.1/0.015 = 206\Omega$$

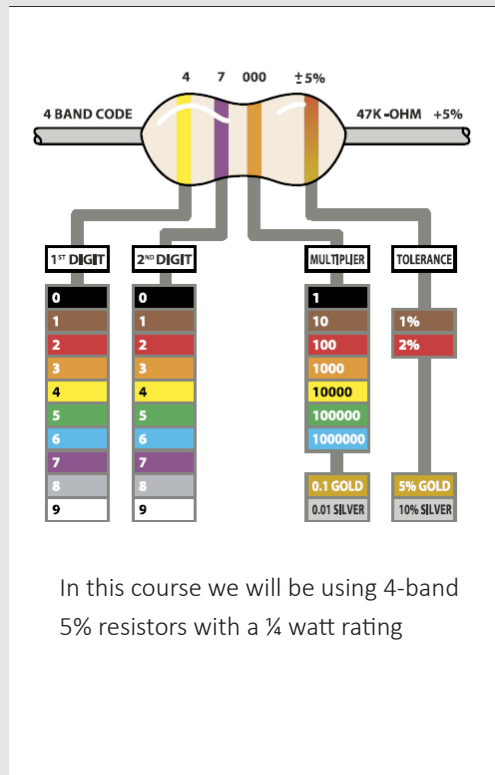
- Exact isn't critical – use next size up, i.e. 220 $\Omega$

- Or be safe and use 330 $\Omega$  or 470 $\Omega$

- This would result in 9.4mA or 6.6mA which is fine



## Resistor Color Codes



What's the color code for a 330 $\Omega$  resistor?

orange orange brown gold

What's the color code for a 1k $\Omega$  resistor?

brown black red gold

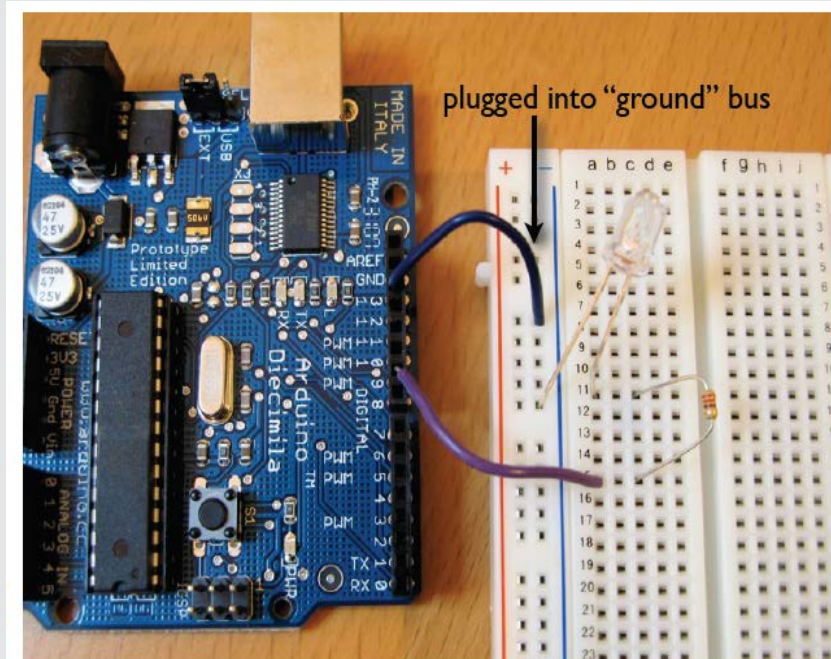
What's the color code for a 470 $\Omega$  resistor?

brown black orange gold

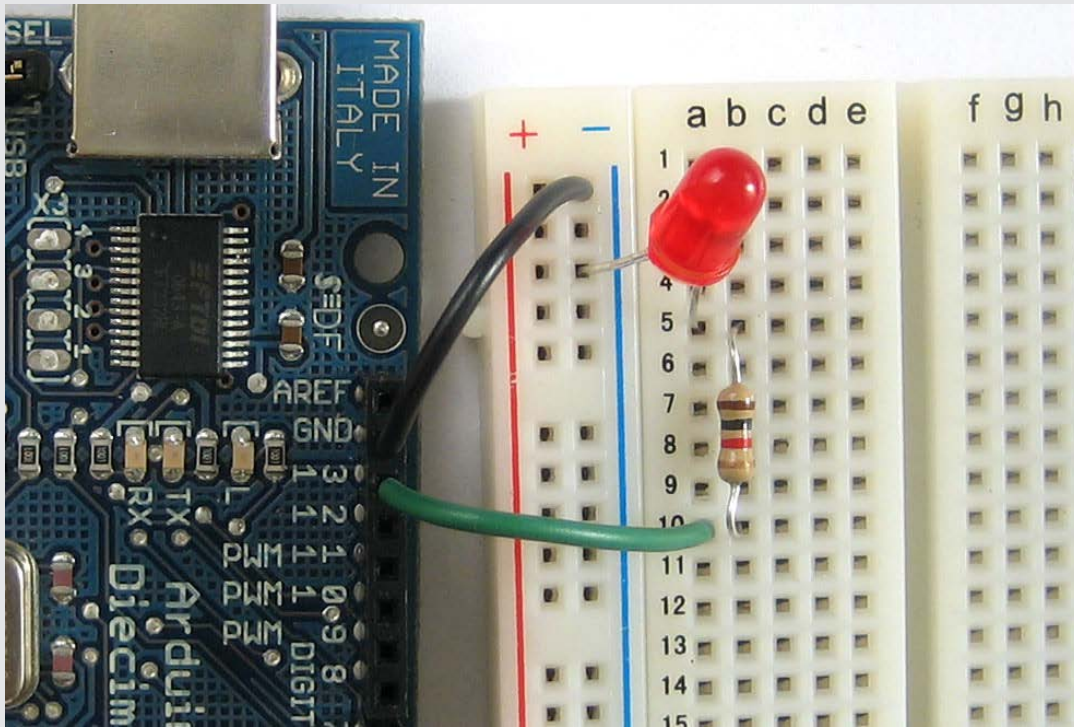
## Wire it up!

Wire up an external LED of your choice, and change the Blink program to use that external LED

- Choose your resistor based on the  $V_f$  of the LED you're using
  - Usually 1.8-2.2v
  - Listed on class web site
- If you don't know  $V_f$  pick 330 $\Omega$  or 470 $\Omega$



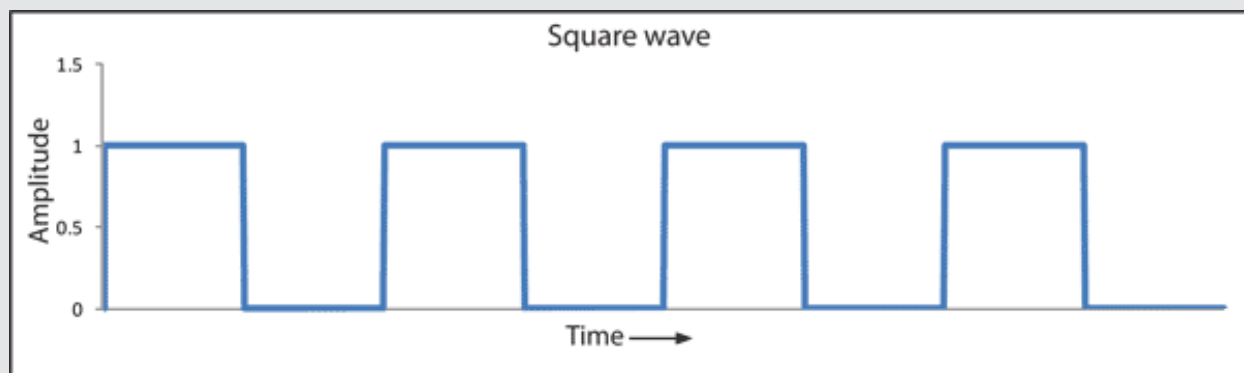
## Another View



## Sound!

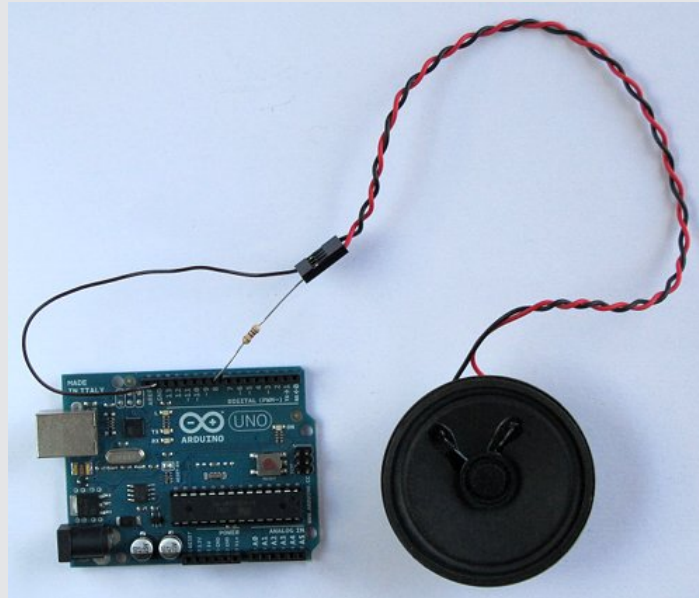
Now – how do we make noise with an Arduino?

- Use a digital output
- Flip it up and down at an audio frequency



## Sound!

Now connect a speaker to that pin...



## Why include a resistor?

- ❑ Ohm's Law:  $V = IR$  or  $R = V/I$
- ❑ Arduino pins can provide, or consume 40mA
  - ❑ 40mA is 0.040A (1 mA is 1/1000 of an Amp)
- ❑ So,  $5V/0.040A = 125\Omega$ 
  - ❑ Or, if you want to be safe,  $5V/0.035A = 143\Omega$
- ❑ Speakers are typically  $8\Omega$ 
  - ❑  $125-8 = 117\Omega$      $143-8 = 135\Omega$

*This is a "current limiting resistor"*

## Make Sound from a Program

The Arduino function that makes a sound is

```
tone(<pin>, <freq-in-Hertz>);  
tone(<pin>, <freq-in-Hertz>, <duration-in-ms>);
```

Examples:

```
tone(10, 440); // play a 440Hz tone on pin 10  
noTone(10);    // stop playing the tone on pin 10
```

```
int myPin = 9; // define a variable named myPin  
int myTone = 440; // another named myTone  
tone(myPin, myTone, 1000); // play for 1sec
```

## Standard pitches – “pitches.h”

Codifies “standard” pitches

```
#define NOTE_FS4 370  
#define NOTE_G4 392  
#define NOTE_GS4 415  
#define NOTE_A4 440  
#define NOTE_AS4 466  
#define NOTE_B4 494
```

```
tone(myPin, NOTE_A4); //  
play standard A above
```

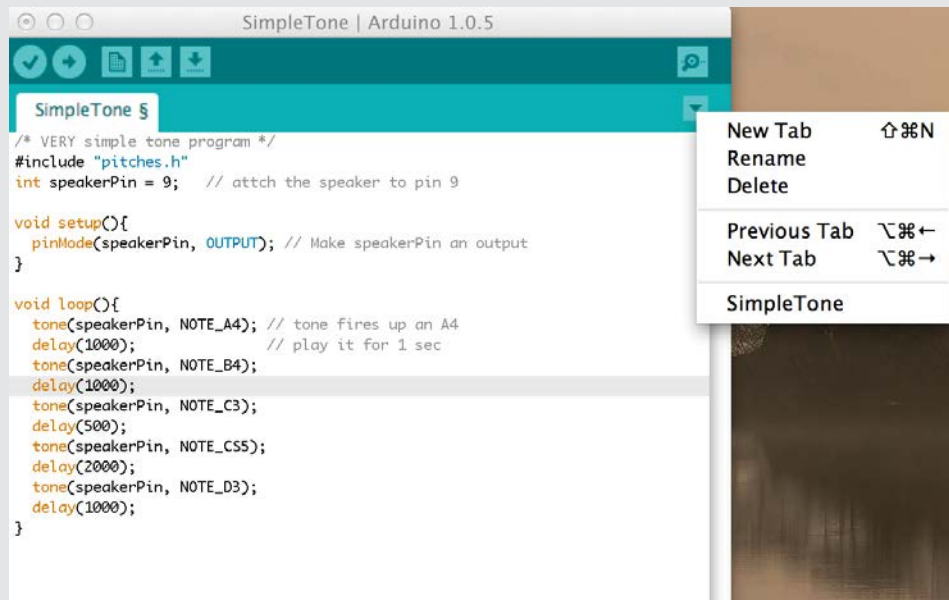
### Example:

```
/* VERY simple tone program */  
#include "pitches.h"  
int speakerPin = 9; // attach the speaker to pin 9  
  
void setup(){  
  pinMode(speakerPin, OUTPUT); // Make speakerPin an output  
}  
  
void loop(){  
  tone(speakerPin, NOTE_A4); // tone fires up an A4  
  delay(1000);                // play it for 1 sec  
  tone(speakerPin, NOTE_B4);  
  delay(1000);  
  tone(speakerPin, NOTE_C3);  
  delay(500);  
  tone(speakerPin, NOTE_CS5);  
  delay(2000);  
  tone(speakerPin, NOTE_D3);  
  delay(1000);  
}
```

## Getting the file pitches.h

- Go to <http://arduino.cc/en/Tutorial/Tone>
- Copy the pitch data
- In the Arduino editor make a new "tab"
- Name the tab "pitches.h"
- Paste in the data

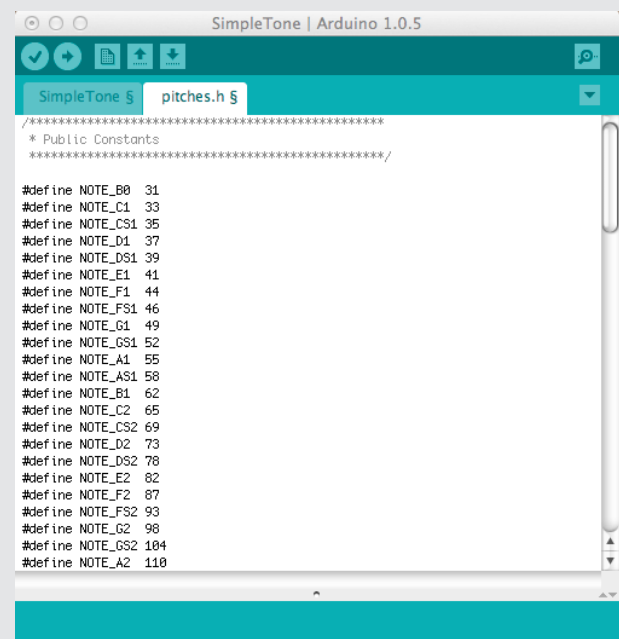
- Making a new tab -->



- name it "pitches"

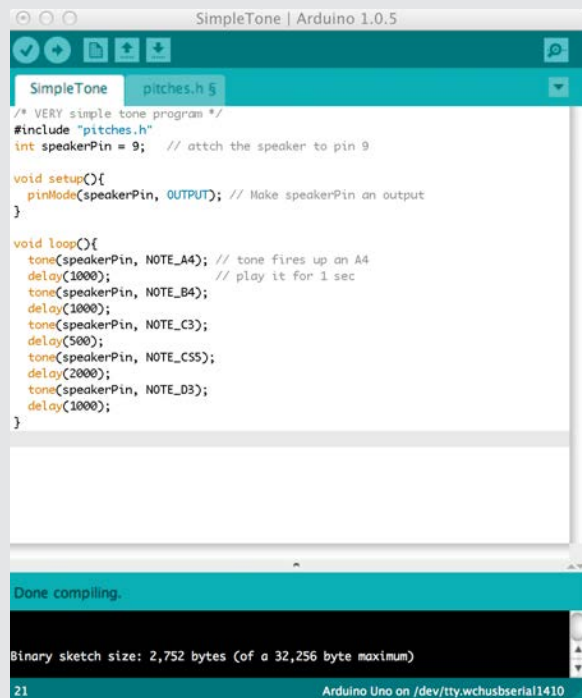


- and add the data from the website.





□ compile and run!



The screenshot shows the Arduino IDE interface. The top toolbar has icons for compiling, uploading, and opening files. The main window displays the 'SimpleTone' sketch with the following code:

```
/* VERY simple tone program */
#include "pitches.h"
int speakerPin = 9; // attach the speaker to pin 9

void setup(){
  pinMode(speakerPin, OUTPUT); // Make speakerPin an output
}

void loop(){
  tone(speakerPin, NOTE_A4); // tone fires up an A4
  delay(1000); // play it for 1 sec
  tone(speakerPin, NOTE_B4);
  delay(1000);
  tone(speakerPin, NOTE_C3);
  delay(500);
  tone(speakerPin, NOTE_CS5);
  delay(2000);
  tone(speakerPin, NOTE_D3);
  delay(1000);
}
```

At the bottom, a status bar indicates 'Done compiling.' and 'Binary sketch size: 2,752 bytes (of a 32,256 byte maximum)'. The bottom-most status bar shows '21' and 'Arduino Uno on /dev/tty.wchusbserial1410'.

□ add space between notes

```
/* VERY simple tone program */
#include "pitches.h"
int speakerPin = 9; // attach the speaker to pin 9

void setup(){
  pinMode(speakerPin, OUTPUT); // Make speakerPin an output
}

void loop(){
  tone(speakerPin, NOTE_A4); // tone fires up an A4
  delay(1000); // play it for 1 sec
  noTone(speakerPin); // stop the tone
  delay(300); // "play" some silence
  tone(speakerPin, NOTE_B4); // play another tone
  delay(1000);
  tone(speakerPin, NOTE_C3);
  delay(500);
  tone(speakerPin, NOTE_CS5);
  delay(2000);
  tone(speakerPin, NOTE_D3);
  delay(1000);
}
```

□ add space between notes

```
/* VERY simple tone program */
#include "pitches.h"
int speakerPin = 9; // attach the speaker to pin 9

void setup(){
  pinMode(speakerPin, OUTPUT); // Make speakerPin an output
}

void loop(){
  tone(speakerPin, NOTE_A4, 1000); // tone fires an A4 for 1sec
  delay(1500); // delay for 1.5sec for some space
  tone(speakerPin, NOTE_B4, 1000);
  delay(1500);
  tone(speakerPin, NOTE_C3, 500);
  delay(700);
  tone(speakerPin, NOTE_CS5, 1500);
  delay(2000);
  tone(speakerPin, NOTE_D3, 1000);
  delay(1300);
}
```

## Additional programming

□ Generate random number

□ `random(<min>,<max>)`

□ Returns random number between min and max-1

□ `random(2, 5);` // returns random number between 2 and 4

## Random

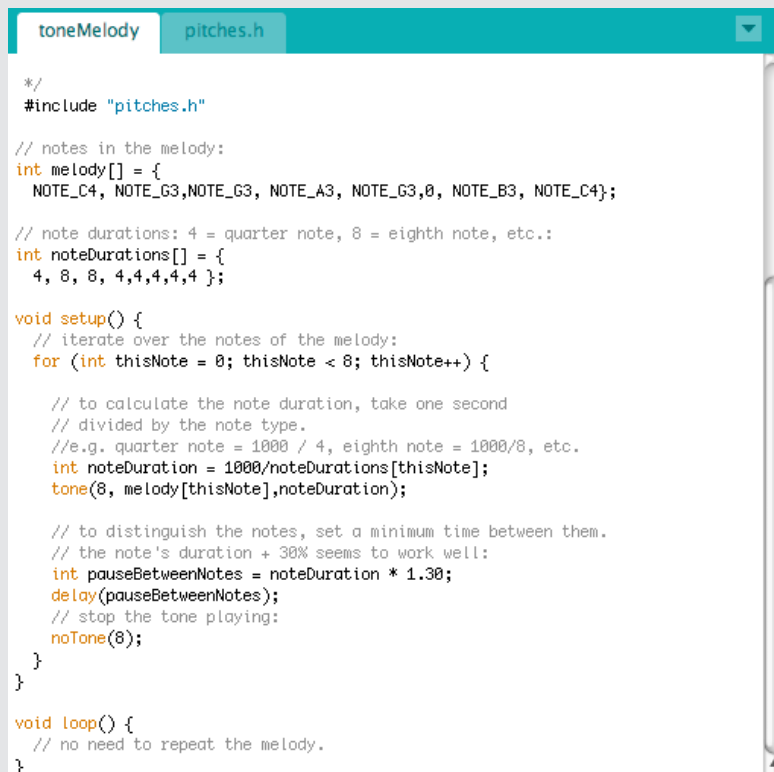
```
int myNum;    // variable to hold a number
myNum = random(1000, 2001); // between 1000, 2000

tone(9, myNum, 1000); // play a random tone

tone(9, random(500, 1500)); // play another random tone

tone(9, 440, random(1000, 2000)); // play for a random duration
```

## Example from arduino



```
toneMelody pitches.h

*/
#include "pitches.h"

// 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(8, 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(8);
  }
}

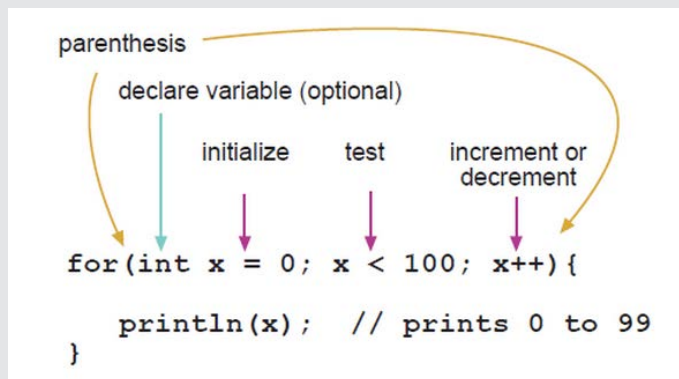
void loop() {
  // no need to repeat the melody.
}
```

## C "for" loop (iteration)

```
for (<initialization>; <condition>; <increment>) {  
  // do something...  
}
```

```
int i;                // define an int to use as a loop variable  
for (i = 0; i < 256; i=i+1) { // repeat 256 times  
  tone(9, i+100);      // play a tone on pin 9  
  delay(1000);        // delay for a second while it's playing  
}                     // plays tones from 100 to 355 Hz
```

## Another view of a "for loop"



### Aside: C Compound Operators

```
x = x + 1;      // adds one to the current value of x
x += 5;        // same as x = x + 5
x++;           // same as x = x + 1

x = x - 2;     // subtracts 2 from the current value of x
x -= 3;        // same as x = x - 3
x--;           // same as x = x - 1

x = x * 3;     // multiplies the current value of x by 3
x *= 5;        // same as x = x * 5
```

### Arrays

▣ Def: A collection of variables accessed with an index number.  
All of methods below are valid ways to create (declare) an array:

```
int myInts[6];
int myInts[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

▣ Indices matter! Consider the following:

```
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
```

▣ To assign a value to an array:

```
mySensVals[0] = 10;
```

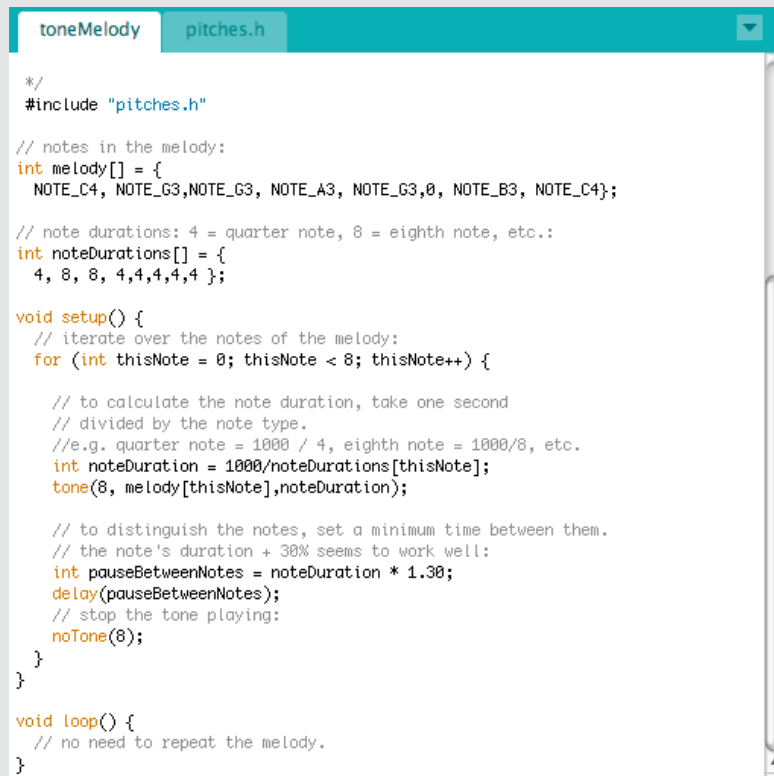
▣ To retrieve a value from an array:

```
x = mySensVals[4];
```

▣ *One more thing to note:* Arrays are often used inside loops:

```
int i;
for (i = 0; i < 5; i = i+1){
    Serial.println(myPins[i]);
}
```

## Back to our example from Arduino



```
toneMelody pitch.h
*/
#include "pitch.h"

// 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(8, 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(8);
  }
}

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

## Summary - Whew!

### □ Digital Pins

- use `pinMode(<pin>, <INPUT/OUTPUT>)` for setting direction
  - Put these in the `setup()` function
  - `pinMode(13, OUTPUT);` // set pin 13 as an output
- use `digitalWrite(<pin>, <HIGH/LOW>)` for on/off
  - `int LEDpin = 10; digitalWrite(LEDpin, HIGH);` // turn on pin "LEDpin"

### □ `delay(val)` delays for val-number of milliseconds

- milliseconds are thousandths of a sec (1000msec = 1sec)
- `delay(500);` // delay for half a second

### □ `random(min, max)` returns a random number between min and max-1

- You get a new random number each time you call the function
- `foo = random(10, 255);` // assign foo a random # from 10 to 254

### □ Two required Arduino functions

- `void setup() { ... }` // executes once at start for setup
- `void loop() { ... }` // loops forever
  - statements execute one after the other inside loop, then repeat after you run out

## Summary cont...

```
int i = 10; // define an int variable, initial value 10
```

### Other types of variables:

- char – 8 bits
- long - 32 bits
- unsigned...
- float – 32 bit floating point number

```
for (<start>; <stop>; <change>) { ... }  
for (int i=0; i<8; i++) { ... } // loop 8 times. The value of i in each iteration is 0, 1, 2, 3, 4, 5, 6, 7
```

```
if (<condition>) { ... }  
if (foo < 10) {digitalWrite(ledPin, HIGH);}  
Conditions: <, >, <=, >=, ==, !=
```

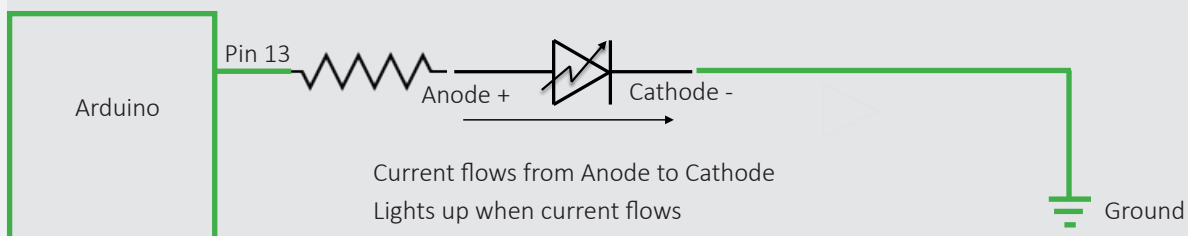
```
if (<condition>) { ...} else { ... }  
if (num == 10) { <do something> }  
else { <do something else> }
```

## Speakers

- If you're going to use a speaker, use a current-limiting resistor
  - Most speakers have  $8\Omega$  of resistance
  - Some are  $4\Omega$  or  $16\Omega$
- Arduino pins can provide or consume 40mA (0.040A)
  - Be conservative – if you're between resistors, use a slightly larger one...
  - $150\Omega$  is a great choice for a speaker

## Last but not least...

- LEDs – turn on when current flows from anode to cathode
  - Always use a current-limiting resistor!
  - Remember your resistor color codes
  - 220-470 ohm are good, general-purpose values for LEDs
  - Drive from Arduino on digital pins
  - Use PWM pins if you want to use analogWrite for dimming

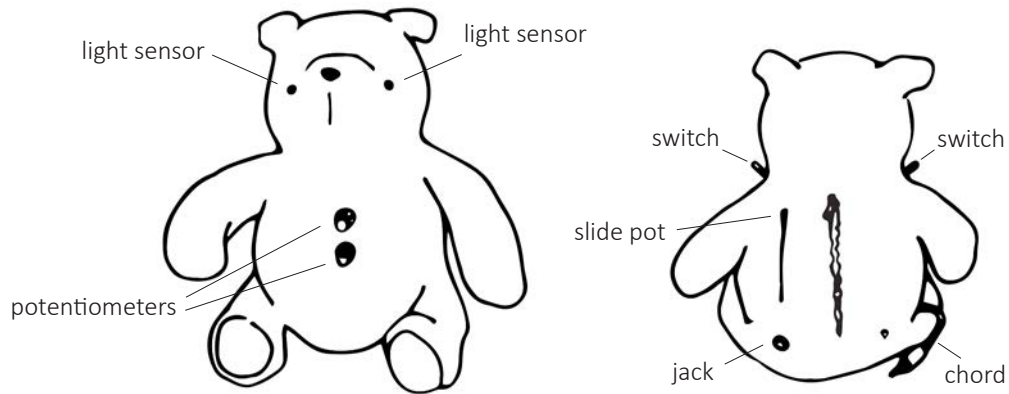




## Resources

- ▣ <http://arduino.cc/en/Tutorial/HomePage>
- ▣ <http://www.ladyada.net/learn/arduino/index.html>
- ▣ <http://todbot.com/blog/bionicaudio/>
- ▣ <http://todbot.com/blog/spookyarduino/>
- ▣ <http://sheepdogguides.com/arduino/aht0led.htm>

# 4 OSCILLATORS



For this assignment you'll be designing and building an oscillator-based instrument. The instrument can be of your own design, but should incorporate at least three oscillators, and use both the CD40106 inverter chip and the CD 4093 NAND chip.

**Reading:** The source material for this assignment is contained in Chapter 18 and Chapter 20 of your text book.

- Chapter 18 talks about using the CD40106 inverter chip to make an oscillator, and how to mix multiple oscillators into a single output. Note that the book calls the chip a 74C14, but this is essentially the same chip as the CD40106 chips that we have, and they behave the same way.
- Chapter 20 talks about using the NAND gate to make a gated oscillator. This means that you can use a switch to turn the oscillator on and off, or use one oscillator to switch, or modulate, another oscillator. It also talks about volume control, and simple filtering. Finally, Chapter 20 also briefly mentions "voltage starving" as a way to increase the complexity of the tones coming from the oscillators.

The basic assignment is to explore these oscillators, and build a small oscillator-based instrument in some sort of enclosure. You'll demonstrate your instrument on Thursday, April 9th in class.

Some things to consider...

- I recommend building at least the oscillators in Figures 18.8, 18.10 and 18.19 to try them out and see how they sound.
- In Chapter 20 I recommend trying the oscillators in Figure 20.4 and perhaps 20.13, and trying voltage starving.
- That should give you a range of options that you can build upon to decide what your own instrument should contain. You are in charge of your own instrument – use the oscillators that you like the sound of.
- You should have a breadboard with one each of the CD40106 and CD4093 chips on them. If you need more chips, let me know.
- You should make your own jumper wires using the solid-core wire that's in the lab. The best practice here is to cut the jumpers to be the right size for where they need to go so that they don't make too much of a

rainbow shape above the board. This way they won't fall out and get caught as easily.

- There are pots and switches in the lab on the lab bench in the plastic bags. You can use slide pots or knob pots, and you can use switches to connect and disconnect oscillators from the circuit.
- There are diodes, resistors, capacitors, and CdS light sensors (variable resistors) in the small plastic cabinet on the north wall of the lab. The drawers are labeled. Please put things back in the correct drawer if you're done with them. Nothing is worse than grabbing a part from a labeled drawer and finding that it's not the right part!
- For your final instrument, put it inside an enclosure of some sort. You can use a cigar box (I put some additional boxes in the lab), or your own enclosure that you get from wherever you like. You could even use an Altoids tin or something like that if you like. Or a tin can. Or a Tupperware. Or a cereal box. Or a plush toy. Or an old shoe. Pretty much anything that you want to pick up and interact with when you're finished.
- If you use a metal box, make sure that you aren't shorting your circuits to the metal box! This may require some care and thinking about how things are organized, and perhaps even some extra shrink tubing, or other insulation.
- If you'd like to solder your instrument together once you've prototyped it, you're welcome to do that. See Chapter 19 for thoughts on soldering a project like this. I can provide small boards to solder on.
- Consider how you're going to get the sound out of your instrument. You'll need to send the signal to an amplifier of some sort. The amplifier you've been using for recording works fine.
- Your basic choices for getting the sound out are to make a plug (either 1/4" or 1/8" ) that you can plug directly into the amplifier (there are plugs with wires poking out hanging on the rack in the lab). Or you can use a jack (again, either 1/4" or 1/8") that you can plug into. In this case, you'll need a cable with 1/4" or 1/8" plugs on both ends (like an instrument cable). We have some of these too.
- Your instrument should have multiple sounds – that is it shouldn't just buzz at one boring frequency. It should be "playable" by manipulating things. The "things" to manipulate can be switches, knobs, sliders, CdS light sensors, body contacts, or anything else that you can think of to influence the oscillator behavior.
- Make your finished instrument look good! I'm looking forward to seeing what you come up with!

*Don't hesitate to ask for advice or suggestions!*

### ***What to turn in...***

On Canvas:

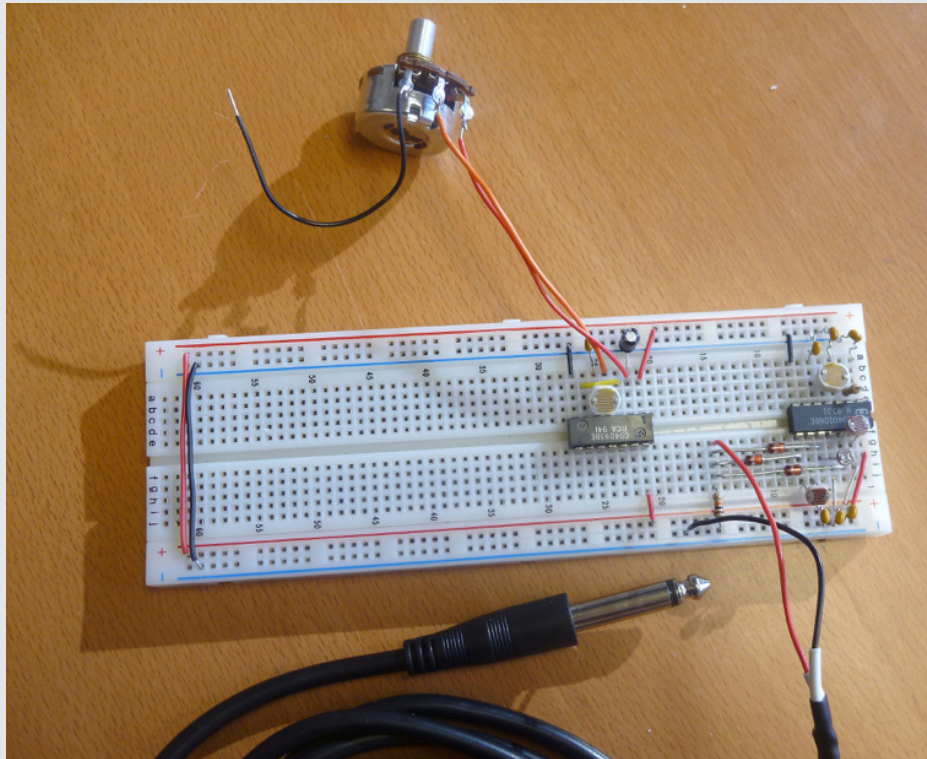
1. Turn in photos of your instrument showing both the inside and outside.
2. Turn in (using SoundCloud) sound samples from playing your instrument.
3. Turn in a one-page guide to using/playing your instrument. Let us know what the controls are, and perhaps even some suggestions for “good” settings of those controls.

In Class:

4. Bring your instrument to class and demonstrate it on Thursday, April 9th.

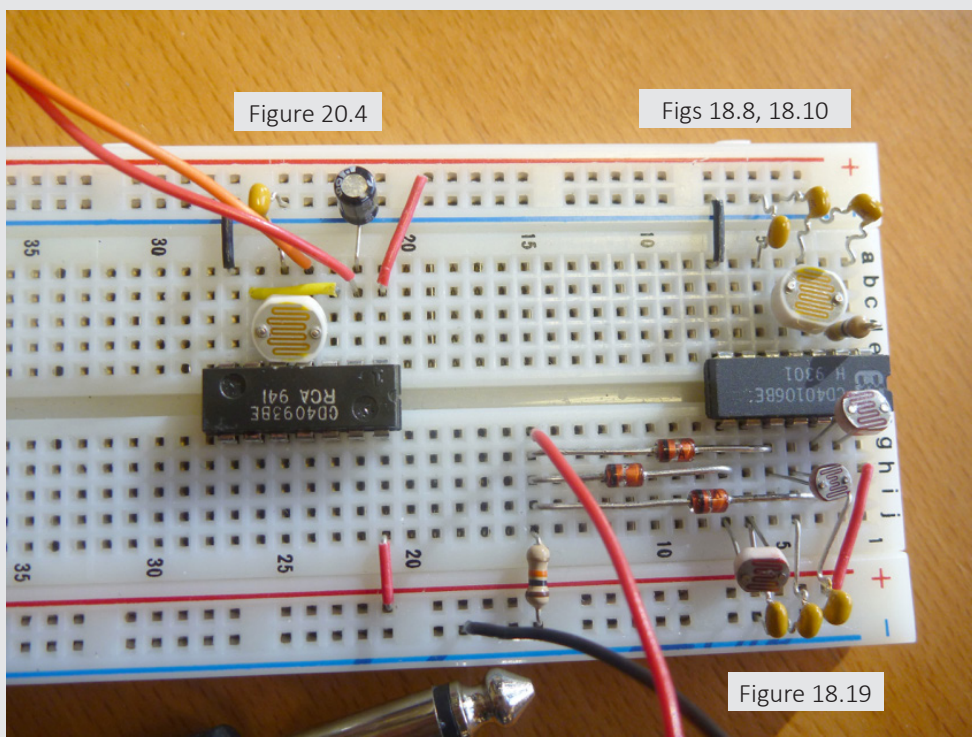
*In the following, we show various components which may be useful for your oscillator*

## Example Oscillator



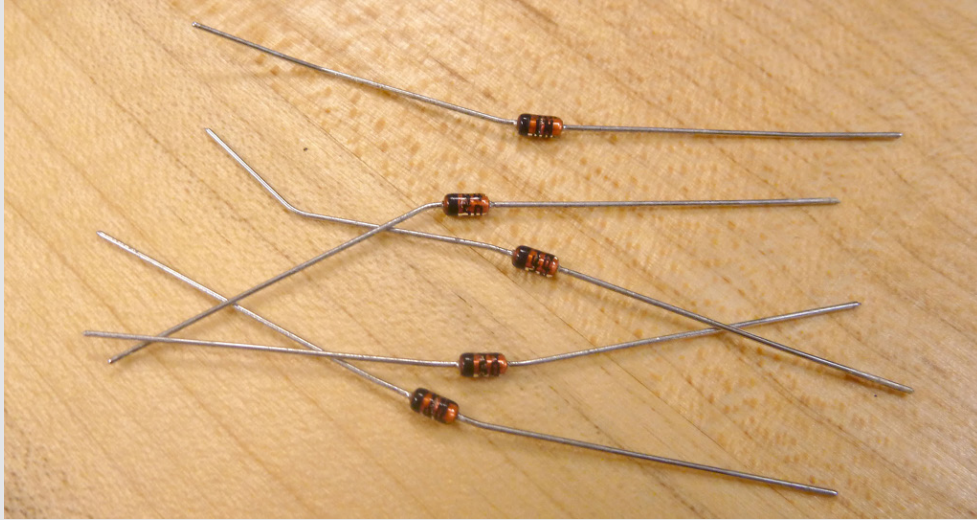
## Both Chips

Please see the corresponding figures in the textbook for more detail.

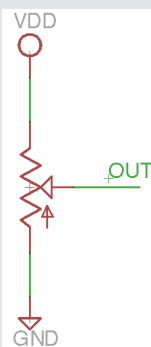
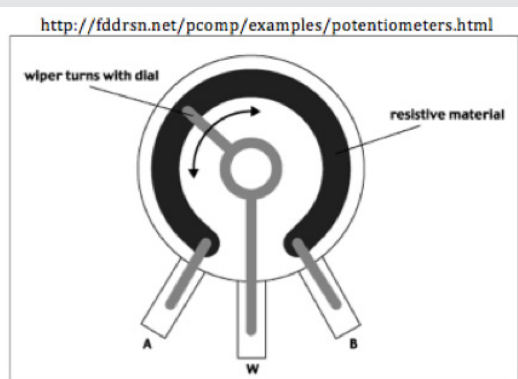




## Diodes

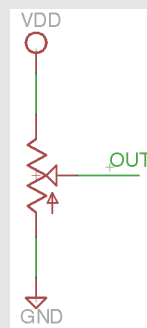
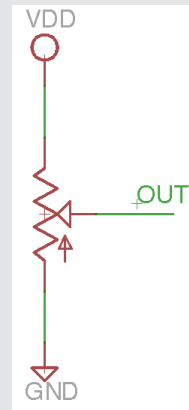
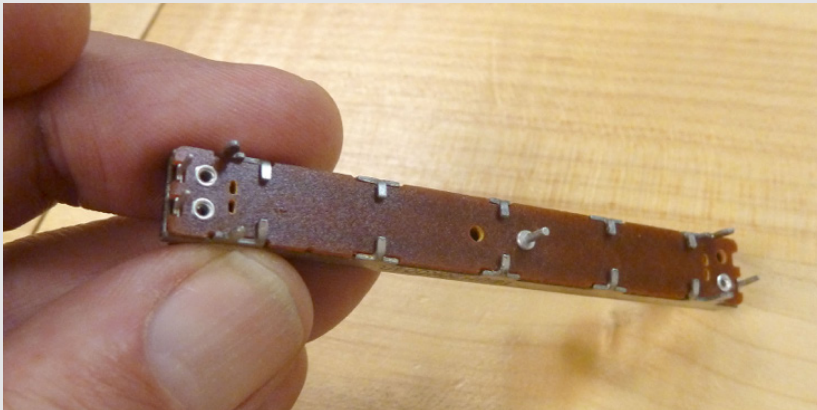


## Potentiometer (a.k.a “pot”)

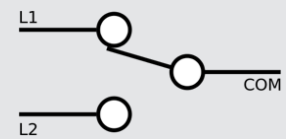
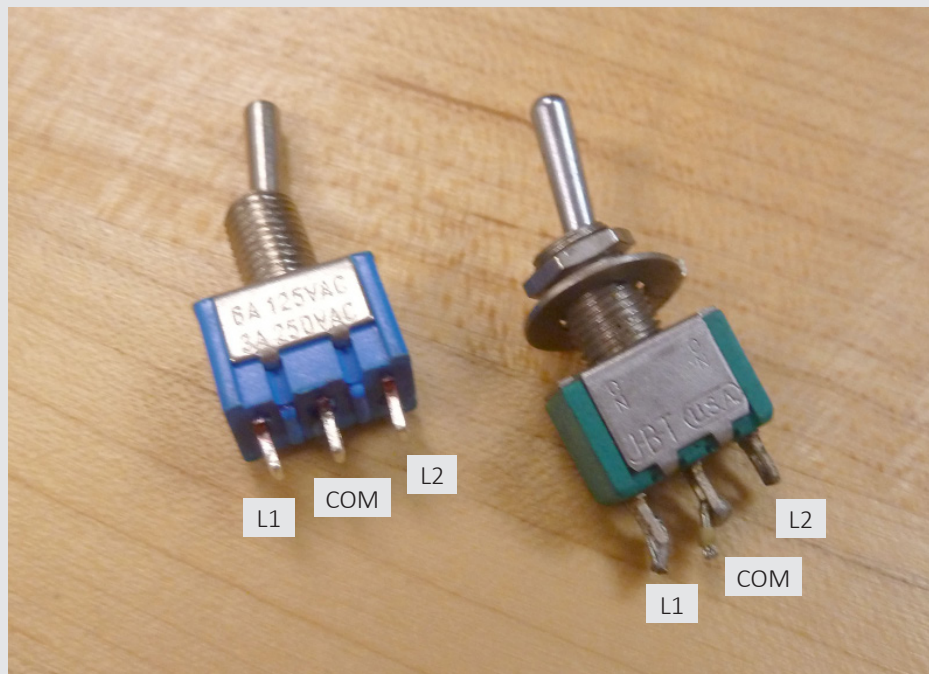




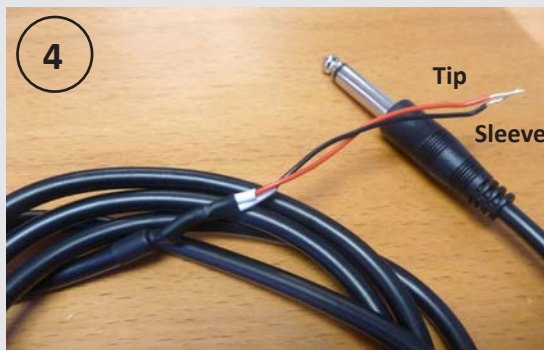
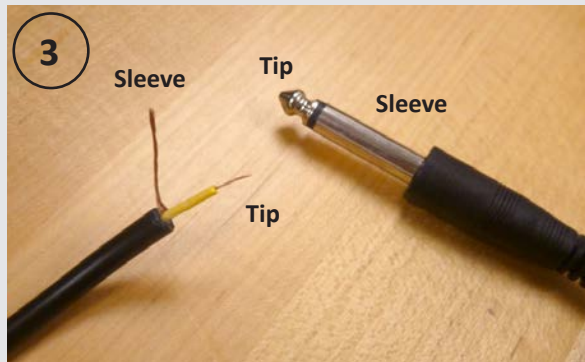
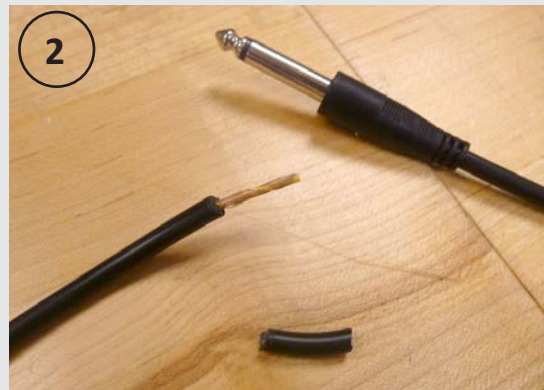
## Slide Pot



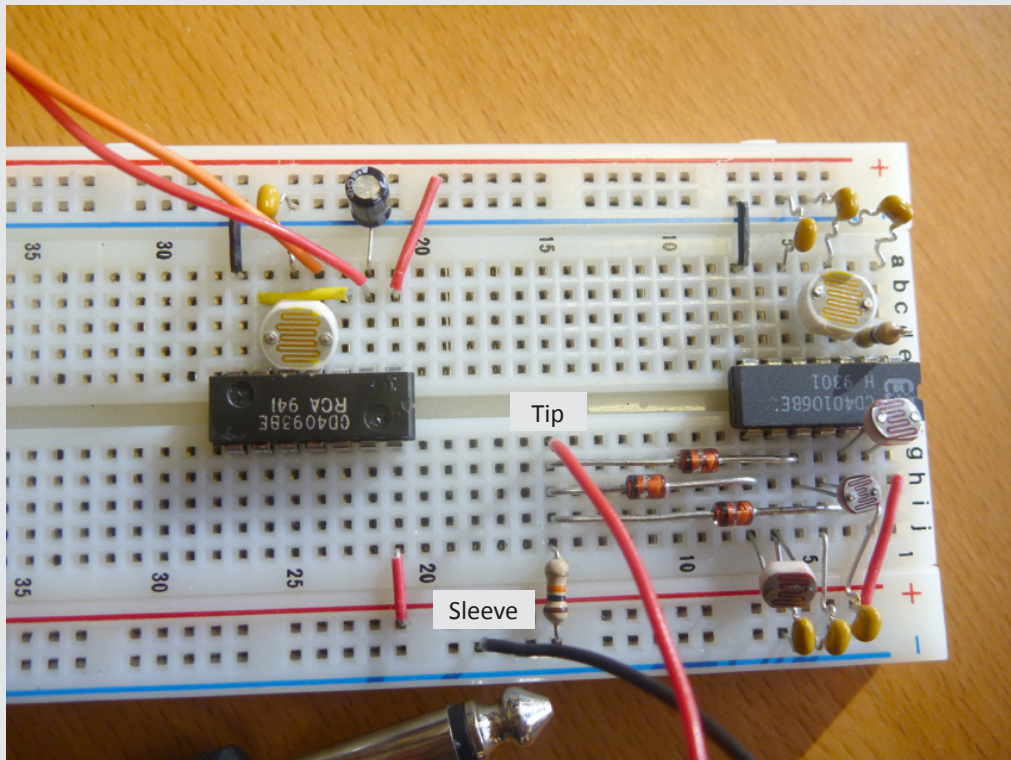
## Switches



## 1/4 Chord - Prepping

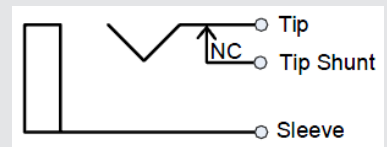
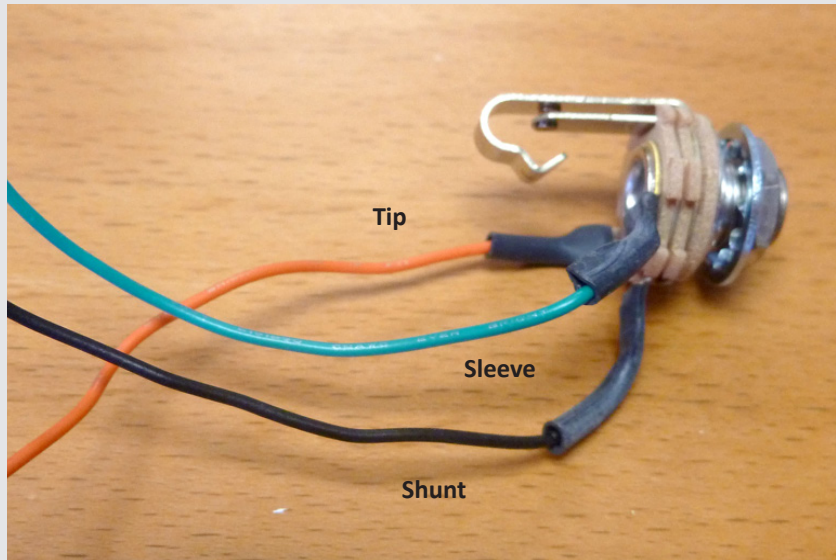


## Adding the chord to the breadboard

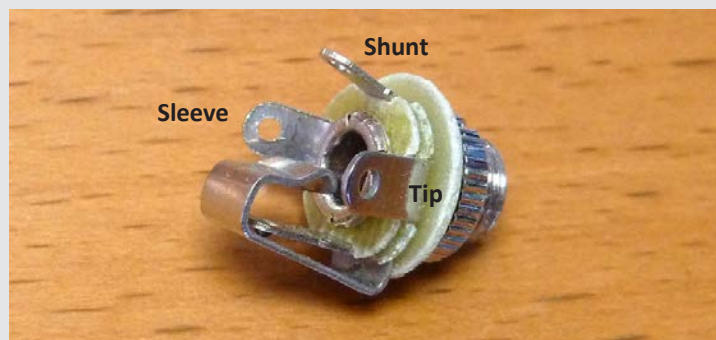
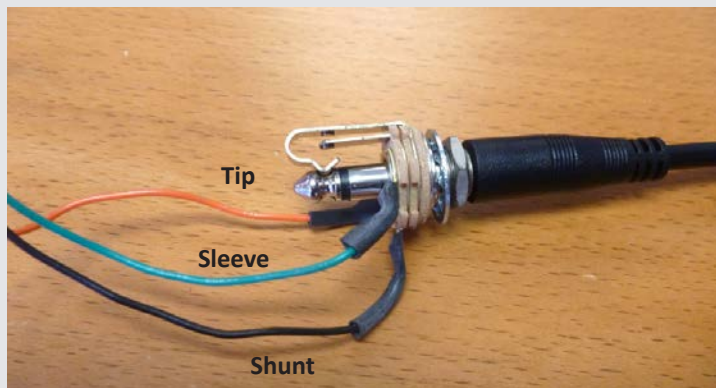




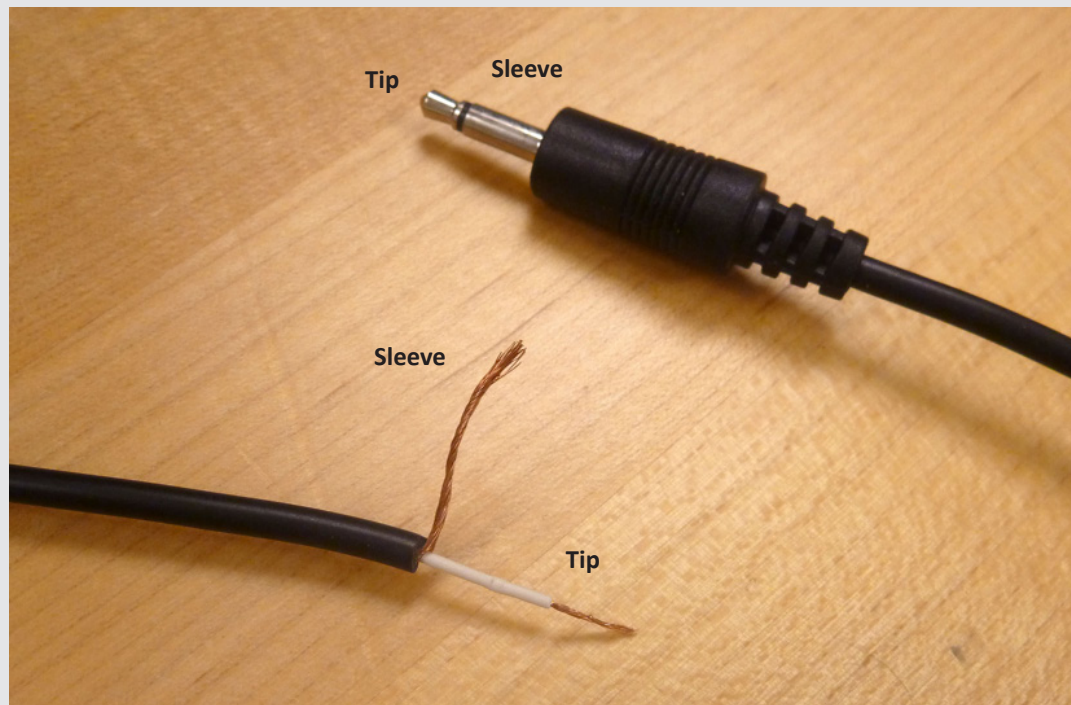
1/4 Jack



1/4 Jack



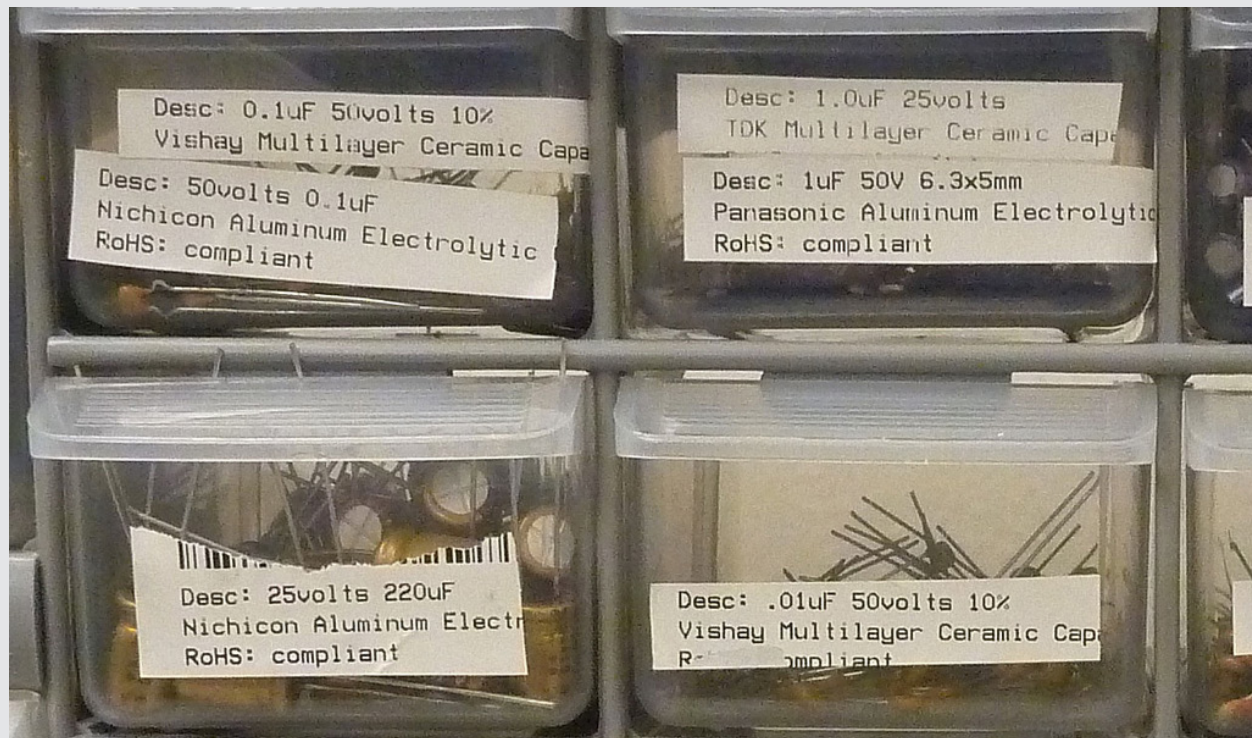
## 1/8 Chord



## Components Storage



## Component Storage





# 7 Final Sound Art Project

For your final project you can use any or all of the raw material that you have collected throughout the semester:

- Inductive coil recordings

- Contact mic recordings

- Arduino sound programs

- Hacked toys turned into strange instruments

- Your own oscillators

We also have other materials that you can use - for example you could build a cigar-box amplifier, use multiple small speakers, etc.

The project is to make some sort of sound art - the specific project and deliverables are up to you. Some examples:

You could compose a longer piece of electronic music using your sound clips as source material

You could make and refine a more complex hacked toy instrument. Along with the instrument you could come up with a “musical” notation that describes how to play the instrument, then compose some music and perform for the class.

You could design and build a sound art installation of some sort. This could be portable where you bring it into class, or it could be site-specific where you install the piece in some other location.

You could come up with some other type of project - it should involve sound/circuits/noise of some sort.

As usual - more details are forthcoming. We will discuss all sorts of project ideas in class before you have to decide on one.

## Project Proposal

Prior to beginning the final project, project proposals must be submitted and approved.

Please submit a one page (or more if you have more to say) document describing your proposed final project. Tell me what you hope to do, what your final demo/performance will be like, what supplies you'll need, and how it builds on the material that we've used in class so far.

### ***What to turn in...***

Proposals may be submitted as a text entry or text document uploaded to canvas.

Details and requirements for the final project will be discussed in class.