Let's Code Blacksburg's Arduino Cookbook

Version .2 2014-02-20

By Monta Elkins, Eddie Sheffield and Thomas Weeks

Online PDF: http://goo.gl/3iL9ZJ

How To Cook (use this Arduino cookbook)

What Is This Cookbook?

This Arduino circuits and programming instruction guide is organized into a "cookbook" style layout. The cookbook illustrates how to create and write various arduino based circuits and programs. These instructions are organized into "Recipes" (instruction guides) that can be combined in different ways to come up with new and different creations (meals? ;) .

How This All Works:

For example, learn how to move a servo motor with our motor recipe. Then learn how to read a knob (aka potentiometer or "pot") with another recipe. Combine them to use the knob to control the servo, or light brightness, or sound. Combine them however you imagine!

The circuit and programming recipes, just like this intro, each feature a **What**, **How** and **Fail** sections – quickly telling you what you're doing, illustrating how to do it, and what to look for if something fails. Identifying failure is important as it is what enables you to "fail fast", learn and move on quickly to success. Failure is good! Without it, learning is difficult and much less satisfying.

Failure:

A bit more on failure – one needs to understand and embrace that *Failure is a natural part of learning*, but before you can learn and move on from failure, you must:

1) *Recognize* failure has or is occurring
2) *Step Back* and determine (or guess at) the nature or root of the problem
3) *Test Your Assumption* (of the problem) and correct your assumption if needed
3) *Address the Problem*, work around it or create a new way of accomplishing the desired outcome.

Also understand that the failure sections should not be looked at as complete guides to every sort of failure you can encounter for a given recipe. No "failure guide" can easily contain every type of possible failure for a given circuit or technical process. The failure section is more a guide to help get your thinking cap back on straight and think about the nature of the problem, and what a quick fix or work around for it might be.

Don't be afraid of failure! Identify it, embrace it, learn from it and move on.
As Thomas Edison says,

> *"Negative results are just what I want. They're just as valuable to me as positive results. I can never find the thing that does the job best until I find the ones that don't."*
> *— Thomas Edison*

As one of our class instructors cries out to his students, "Fail fast, fail cheap!"

Cookbook Recipes:

Recipe Name	Description
*Installing Arduino Sware & Drivers Recipe	Needed on a fresh PC to talk to the Arduino.
*TEST: LED (light) Blink Recipe	Test compiles and uploads code to blink a built in LED.
OUTPUT: Serial Monitor Recipe	Outputs text back to the PC. Great for live troubleshooting.
*BUILD: Breadboard or Protoboard Recipe	Breadboards are your pallet for creating temporary circuits.
INPUT: Potentiometer Recipe	A knob or “pot”, a adjustable resistor for creating a variable voltage.
**BUILD: LED Chase Light Recipe	Wire up and blink a chase light circuit.
CONTROL: Servo Recipe	A servo is a PWM, digital motor that you can control the angular position of.
**OUTPUT: Sound Generation with Arduino	Making tones, sounds and music with speaker.
**INPUT: Light Sensor	Detecting light intensity with a CdS photo-sensor.

* - Required for newer/beginner folks

** - Really fun and easy for newer/beginner folks (light sensor is easier, quicker and more fun than the chase light recipe. But that's just my opinion ;)

NOTE: Any of these recipes can be combined and used together.
For example, you can mix and match the recipes to have:

- the servo can turn the knob to play a sound
- the blinking light recipe can blink into the light sensor and control a motor (or play a sound)
- the potentiometer (knob) + servo recipe together can wave a popsicle stick over the light sensor to make music and tones.
- Multiple servos and potetiometers together can control a two axis robotic, popsicle stick arm (more advanced project for non new folks. Ask for a special recipe handout if interested)

The only limits are your own imagination (and our class duration) :)

Installing Arduino Software & Drivers Recipe (needed on a fresh PC to talk to the Arduino)

What:

This is the process that installs two things; the IDE or programming environment that you use to program and upload code to the Arduino, and the USB/serial port drivers (if needed on Mac and Windows) to "talk" to the board over the USB interface.

> **NOTE:** If you can not get this working (test it using the Blink Recipe), then you will not be able to work with any other recipes in this cookbook.

How:

For Linux

- RedHat:

```
# yum -y install arduino  #(reqs: uisp avr-libc avr-gcc-c++ rxtx avrdude)
```

- Ubuntu:

```
$ sudo apt-get install arduino      #(reqs uisp avr-libc gcc-avr avrdude librxtx-java)
```

- or for other installs or source based installs, go here:
  http://playground.arduino.cc/Learning/Linux

For Mac/OSX:

- Download & Install Software from: http://arduino.cc/en/Guide/MacOSX#.UwGmXXWqYY0

For Windows:

- Download & install the software from: http://arduino.cc/en/Guide/Windows#.UwGlyHWqYY0

Fail:

Linux T-Shooting:

- check permissions of /dev/ttyUSB0 or /dev/ttyACM0 (user needs r/w or 777 access)
- May have to open port permissions to:

```
# usermod -a -G uucp,dialout,lock $USER
```

  or may have to tempfix as root :

```
# chmod 777 /dev/ttyUSB0
```

Windows T-Shooting:

- Make sure the special USB serial port drivers are installed
  http://arduino.cc/en/Guide/UnoDriversWindowsXP#.UwGbtHWqYY0
- Check/fix COM port settings

Mac T-Shooting:

- Make sure the special USB serial port drivers are installed
  http://arduino.cc/en/Guide/MacOSX#toc3
- check device permissions (similar to Linux)

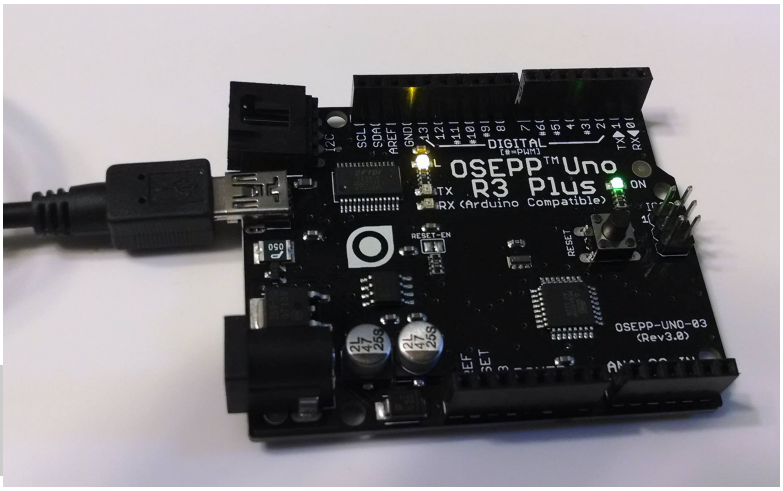
TEST: LED (light) Blink Recipe (test compiles and uploads code to blink a built in LED)

What:

This process simply compiles and uploads code to the Arduino for execution and blinks a light when it succeeds. It's the easiest and fastest (fail fast) test method to verify you can talk to your arduino.

There is a small LED (the light) connected to pin 13 of the Arduino. When that pin is 'high' (meaning +5 volts for this Arduino clone), the LED lights.

> **NOTE:** Arduino Software and driver should be installed. (see Installing Arduino Software and Drivers recipe).



Arduino Clone with power and "blink" light

How:

Select appropriate port
"Tools / Serial port" (see Fail section if the Serial port menu is ghosted).

Select Arduino Uno.
"Tools / Board / Arduino Uno".

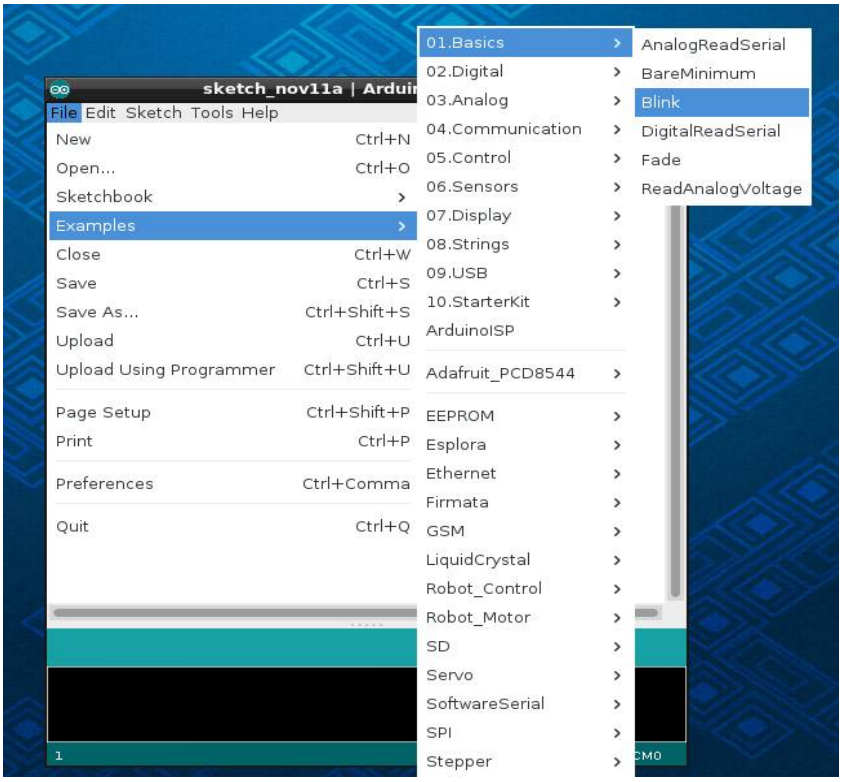
Load blink onto the arduino, "File / Examples / Basics / Blink" (see right).

Click on the upload icon to compile and upload your program.

Several lights will blink during the upload, then your program runs.

Look for the steady light blinking here.

Change both lines that say **`delay(1000);`** to **`delay(100);`** and reupload the program.



Loading the "blink" program in the Arduino IDE

Look for the light to blink faster. This shows that the program changes you made are actually uploaded to the Arduino.

This blinking is done by the command **`digitalWrite(led, HIGH);`** which sends a “HIGH” 5volts to the pin# in the variable “led”. When pin 13 is 'LOW' (meaning connected to ground) the LED (light) is off. This is done by the command **`digitalWrite(led, LOW);`**

> **NOTE:** Anything following “//” on a line is considered comments and ignored.

Fail:

If you can not write to the arduino, get some error, or the serial port is ghosted:
Verify the correct serial port
Unplug Arduino and list serial ports,
then plug up Arduino and list serial ports again. An additional serial port should appear. That new serial port should be the Arduino port.
Try plugging Arduino into a different port USB port
See the fail section of the “Installing Arduino Software & Drivers” recipe (port/permissions/drivers)
Test with a different cable and Arduino board.
Try running the arduino program as root (gets around all permission errors. For testing only)

OUTPUT: Serial Monitor Recipe (outputs text back to the PC. Great for live troubleshooting)

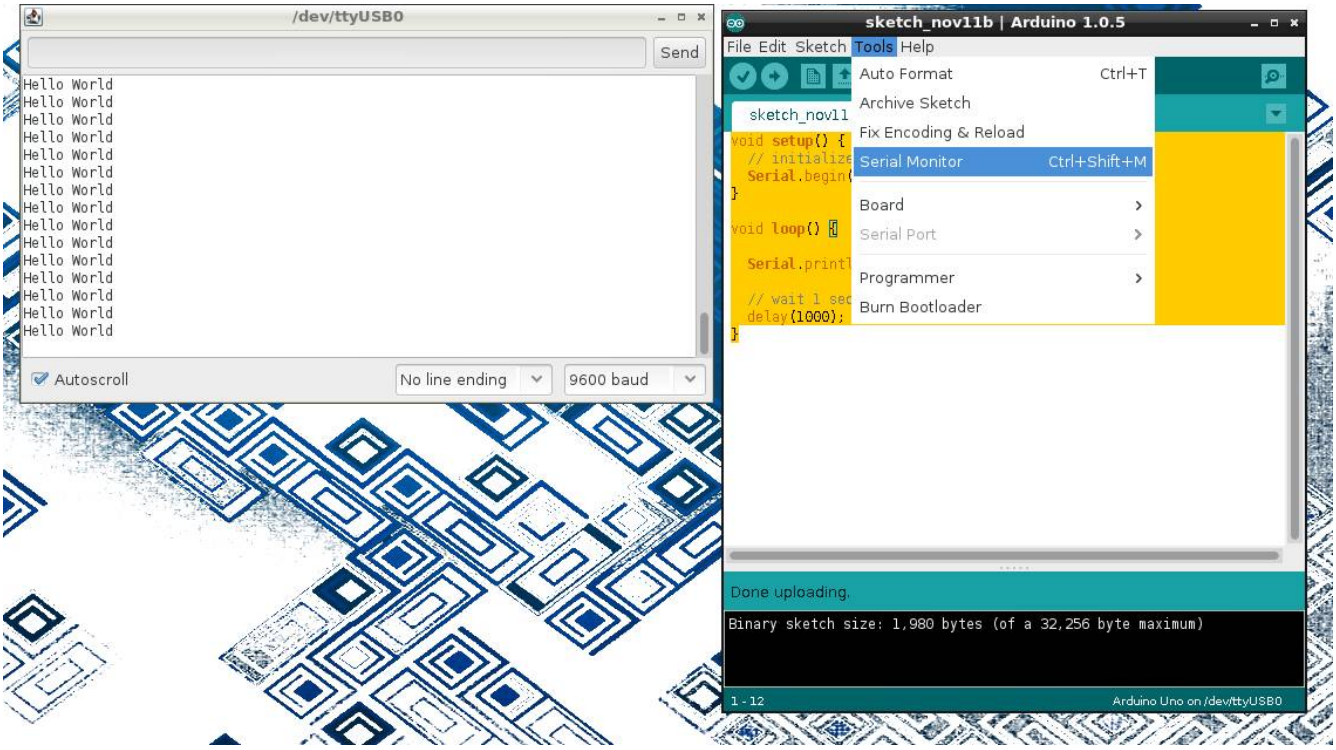
What:

The serial output on the arduino can be used to echo or print real time program data back to the PC over the serial USB port. This is very handy for troubleshooting or looking at run time values, states and problem code.

How:
Upload and execute the following sketch on the Arduino.

```
void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {

  Serial.println("Hello World");

  // wait 1 second before the next loop
  delay(1000);
}
```

Then open the Serial Monitor by selection “Tools / Serial Monitor”.
You should see the line “Hello World” appear repeatedly in the window.



`Serial.begin(9600);` opens the serial port at on the Arduino and sets its speed to 9600 baud
The **`Serial.println()`** statement prints a line followed by a newline.
When the Serial Monitor is opened it looks for characters appearing on the laptop serial port and prints them in the window.

Fail:

- Be sure that you have the **`Serial.begin(9600);`** defined in the **`void setup () {`** code block.
- Look for RX and TX lights to blink on Arduino rapidly during upload.
- If program is running successfully look for the TX light to blink once per second, showing that it is "trying" to transmitting data back to the laptop.
- Double check your Serial Monitor settings for both port and speed.

> **NOTE:** you have to restart the Serial Monitor after each upload of a new program, because the upload process uses the same serial port connection.

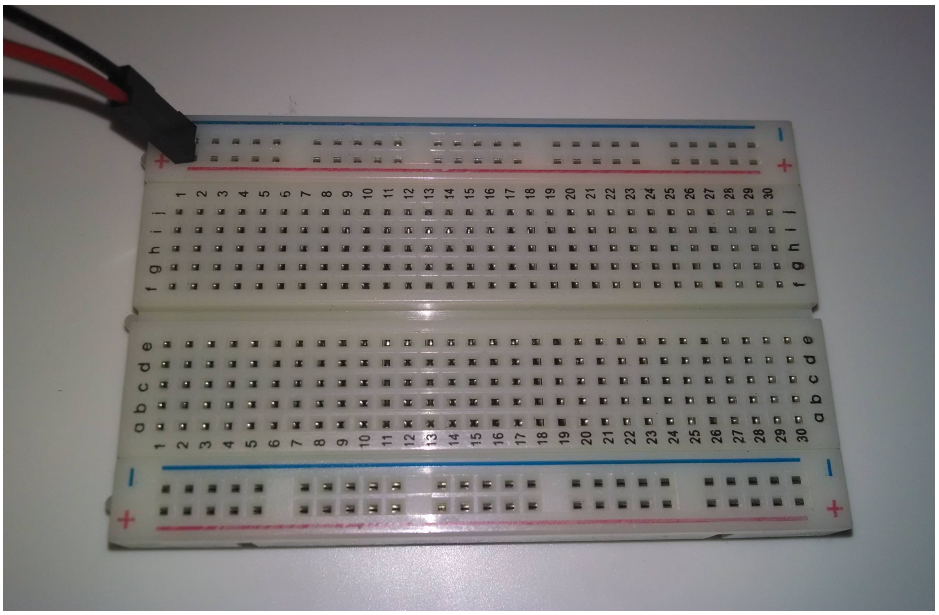
Breadboard or Protoboard Recipe (breadboards are your pallet for creating temporary circuits)

What:

How does a breadboard work?
Wire pins, pushed into the breadboard are connected together as shown in the schematic below. This allows the quick building and testing of circuits.

> **NOTE:** Unplug the Arduino from the laptop (and any other power supply while making and verifying connections).

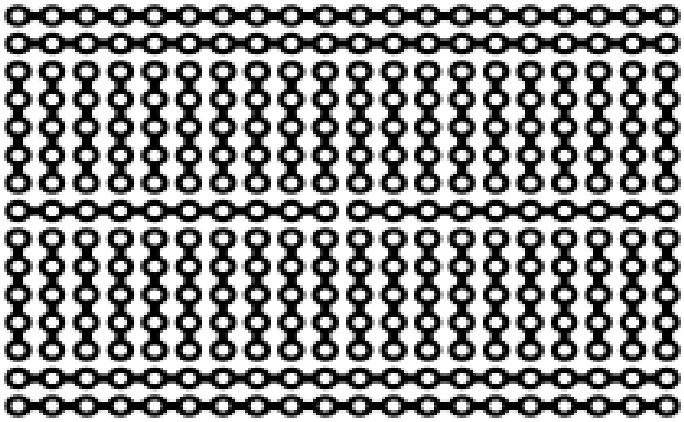
Connection wire colors do not matter; but traditionally power (+) wires are red and ground (-) wires are blue (or black). The (+) red breadboard row and (-) blue breadboard row gives you a common hookup location on the breadboard for power (red) and ground (blue, GND). Using them also makes troubleshooting a little easier.



Average "breadboard" or "protoboard" (w/ horizontal +/- power rails)

Under the white plastic of the breadboard you see that the holes are connected. This is what makes working with a breadboard like legos for electronics.

WARNING: Never connect or short +5v power or (+) to GND or (-). This short will probably blow your laptop's USB port and damage other hardware. Not to mention upsetting your instructor and being mocked by your classmates – probably getting you a lame nick name like “smokey” or “shorty”.

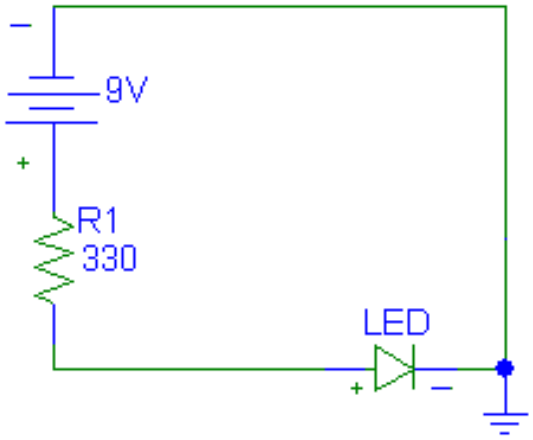


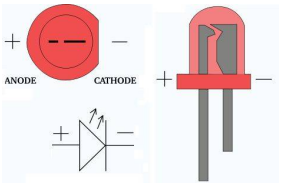
Protoboard schematic. -Wikipedia

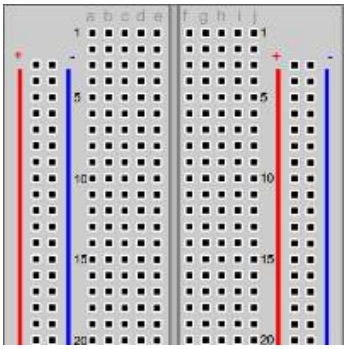
How:

To build a circuit, you normally:

1) start with a schematic diagram (left) which illustrates what is connected to what,
2) orient your components correctly (note the polarity (+ and -) on the LED light)
3) and connect them together on the breadboard:

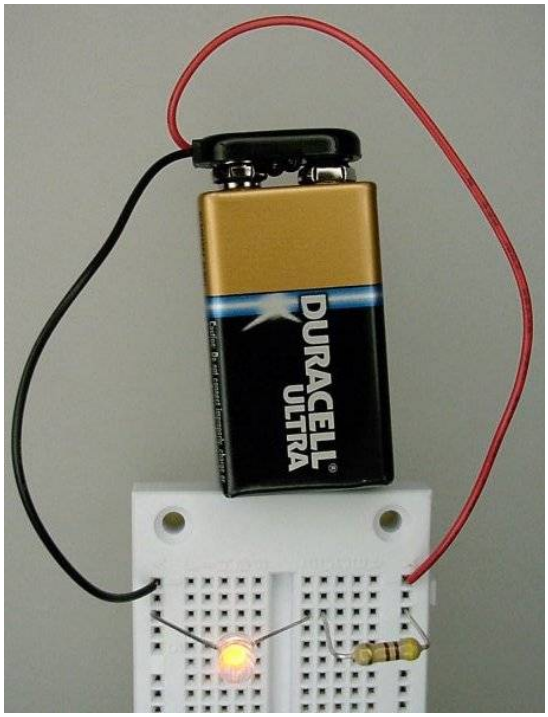






Schematic diagram *LED polarity* *Blank breadboard*

This is what the assembled circuit might look like:



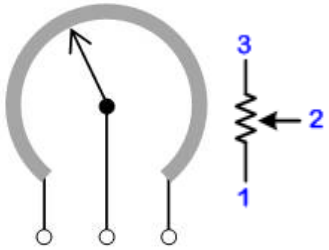
-Batt to ground/-rial to -LED, +LED to Resistor to +rail to +Batt

Fail:

NEVER connect + to – directly. This will damage the breadboard, the wires, the battery and make your system inoperative.

INPUT: Potentiometer Recipe (a knob or “pot” is a adjustable resistor for creating a variable voltage)

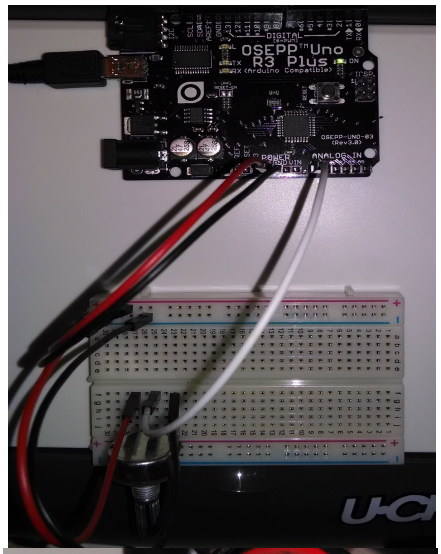
What:

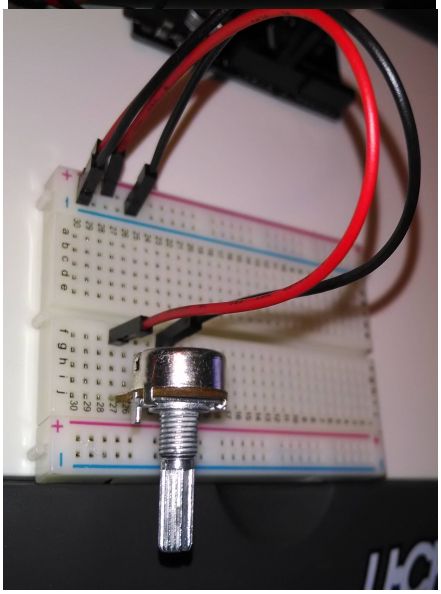


Mechanical and schematic diagrams for a potentiometer

The potentiometer is a just a variable resistor. In most implementations here you will see it used as a knob that will give you a variable voltage (e.g. 0 – 5 volts) to control things hooked to the arduino. The outer two pins, left(1) and right(3), get hooked to ground (GND) and +5 volts, and in this configuration the middle pin (2) will provide a 0 - 5 volt range that can in turn be applied to an arduino analog input such as A0. If operated like this, the arduino will read that value in (when instructed) and convert any analog voltage (at that moment) to a number between 0 - 1023 through a process called analog to digital conversion, much like an MP3 recorder does for audio.

How:





Firmly insert the potentiometer (also called a “pot”) into the breadboard. Connect the leftmost side to ground (GND or -) and the rightmost side to power (+5v or +, or +3.3v if configured for 3.3v operation). The middle connector is the output of the pot in this case. Connect the middle wiper arm to an analog in pin on the Arduino. The A0 input is good.

Verify the circuit by reading and printing the pot value.

```
void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}

void loop() {
  int sensorValue0;
  sensorValue0 = analogRead(A0);  //read the pot input
  Serial.print ("Pot 0 value=");
  Serial.println(sensorValue0);

  // wait 1 second before the next loop
  delay(1000);
}
```

Turn the pot left and right, the printed value should go from (near) 0 to (near) 1023

Fail:

- Verify the pot is seated firmly in the protoboard
- Verify 5 volts across the pot's outter pins with a multi-meter
- Are you reading the input value into a variable?
- Are you printing the correct variable for testing?
  Discuss 'deadband' and “print on change”

BUILD: LED Chase Light Recipe (wire up and blink a chase light circuit)

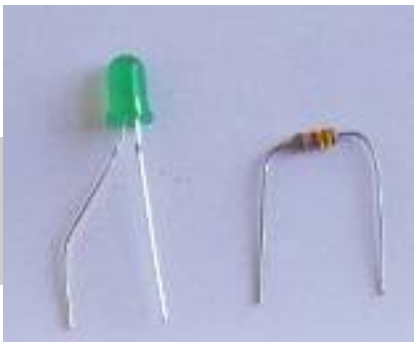
What:

The goal of this recipe is to learn how to hook up multiple LEDs and resistors to the arduino in order that make them strobe back and forth to form a chase light. Add in an optional potentiometer and you can control things like LED chase speed or LED brightness.

You will need:

- a breadboard
- 6-7 LEDs
- a 330 ohm resistor
- 5 or 10k potentiometer (optional)

How:



Before hooking anything up, first note that the LED has a longer leg (positive) and a short leg (negative). Unlike a light bulb, LED lights have + and – polarity.

> **WARNING:** If you get the polarity of an LED hooked up backwards it simply won't light. However, get the polarity right but *without a current limiting resistor* and you can blow it. Please don't blow our LEDs or we'll call you "smokey". :^)

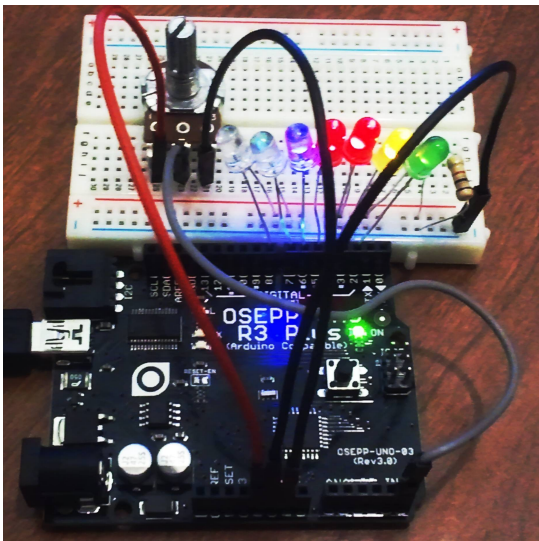
Single LED:
First, start off by hooking up just one LED and current limiting resistor and get that working on the arduino's digital output pin 2. To do this:

- Connect the GND on breadboard (- or blue row) to GND in the "Power" section of the Arduino (bottom/center).

  > **NOTE:** Leave +5V disconnected from the breadboard for this recipe. We're using the + (red) row for something else in this circuit.
- Connect LED's positive (long) leg to digital pin2
- Connect LED's negative (short) leg to the resistor (in breadboard)
- Connect other side of the resistor to GND (blue row) on breadboard
- Load the "blink" program and change **`digitalWrite(13);`** to pin "2"
- Upload & run

Chase Lights:
After you have one LED up and running, hook up the remaining 5-6 LEDs the same way to pins 3, 4 , 5, 6, 7 and 8, all through the same resistor through the red(+) power strip. See photo (right).
Hook up:

- LED positive (long) legs to digital pins 2,3,4,5,6,7,8 (top "DIGITAL" input/output section)
- LED negative (short) legs to breadboard connected together (can use red/positive rail as a common connection point)



- Resistor to all LED negative legs common point
- LED-Resistor's other side to GND (blue rail, wired back to “POWER” GND pin (bottom))
- Modify program **`void setup()`** section to configure LED pins 2-8 as outputs
- Compile and upload 

INSTRUCTOR: Testing, mentoring, t-shooting

Use Pot Value For Timing:

- Add potentiometer to breadboard for variable chase light speed
- Wire pot leg pin 1 (left pin) to ground (GND or 0v) (on breadboard or to arduino GND)
- Wire pot leg pin 2 (middle) to A0 or "Analog0" (on “ANALOG IN” header (bottom right))
- Wire pot leg pin 3 (right pin) to +5v on arduino (on “POWER” header (bottom middle))
- Use "analogRead(0);" in blink program to sample the pot value (from 0 to 1023 max)
- Replace blink's

```
delay(1000);
```

  in milli-seconds with:

```
delay(analogRead(5));
```

  to use the pot read 0-1023 value as the new delay value between LED flashes.

- Compile, upload and run code
- Twist knob to adjust chase light speed (delay)

Try This: You can either read the analogRead(0) just once at the beginning of your LED flashes (using it for each LED on/off cycle), or re-read the pot for each LED flash cycle. Try it both ways. Observe the difference.

Try This: See data from the arduino on your PC in real time with **`Serial.print(analogRead(0));`** to see your pot value in the serial console

Fail:

- LED is not lighting: Check the polarity or try new LED (another “smokey” may have used it ;)
- Pot does nothing: Make sure you have the pot's pins hooked up to GND and +5v correctly or that you're reading the correct analog input.

CONTROL: Servo Recipe

(a servo is a PWM, digitally driven motor that you can control the angular position of)

What:

Connect and control a servo motor (a digitally, position controlled motor) with the Arduino with a potentiometer (knob). You will read the value of the pot, and based on that value, change the position of the servo motor using one of the arduino's PWM (pulse width modulation) outputs.

How:

Use a 3 pin header to connect the servo to the protoboard.

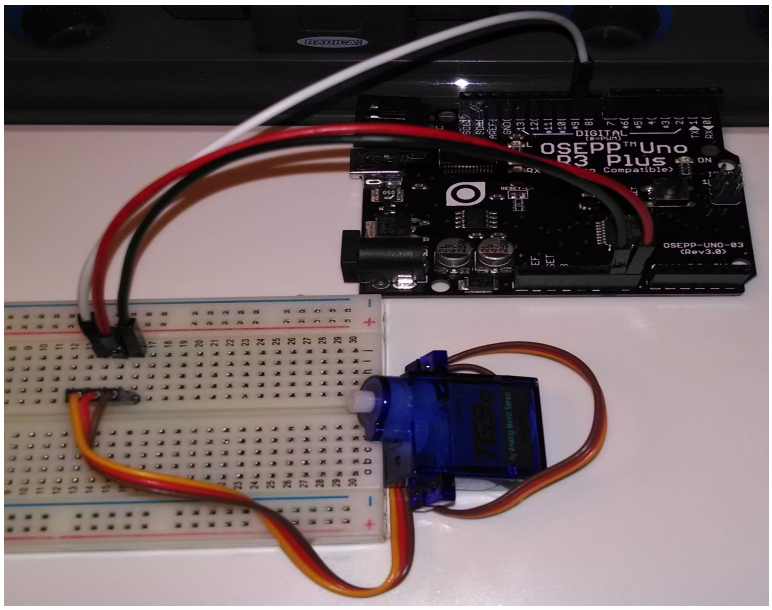
- Connect the brown wire on the servo to ground.
- Connect the red wire on the servo to the Vin pin on the Arduino.
- Connect the orange wire to pin 9 on the Arduino.

WARNING: Placing a 330 ohm buffering resistor on the servo input line (orange) is a good idea to help protect the circuit in case of mis-wiring. "Only you can prevent arduino fires Smokey."

Use the following code to test the servo.

```
#include <Servo.h>
// create a servo object
  Servo servo0;

void setup() {
  servo0.attach(9);  // servo is attached to pin 9
}

void loop() {

  servo0.write(60);  // tell servo to go to the 60 degree position
  delay(1000);       // wait 1 second
  servo0.write(120); // tell servo to go to the 120 degree position
  delay(1000);       // wait 1 second

}
```

Look for the preceding code to move the servo to the "60 degree" position, wait 1 second, then move the servo to the "120 degree" position. The actual movement degree may vary somewhat depending on the servo. The initial position of the plastic servo arm that presses onto the servo toothed shaft may be changed by gently pulling it up, off the servo, turning it and then pressing it back down, re-engaging the "teeth" in a different rotational position.



How:

The position of a servo is set by sending it a 1 – 2 millisecond pulse. A 1ms pulse represents approximately 0 degrees of servo rotation. A 1.5 ms pulse represents approximately 90 degrees. A 2 ms pulse represents a servo position of 180 degrees.

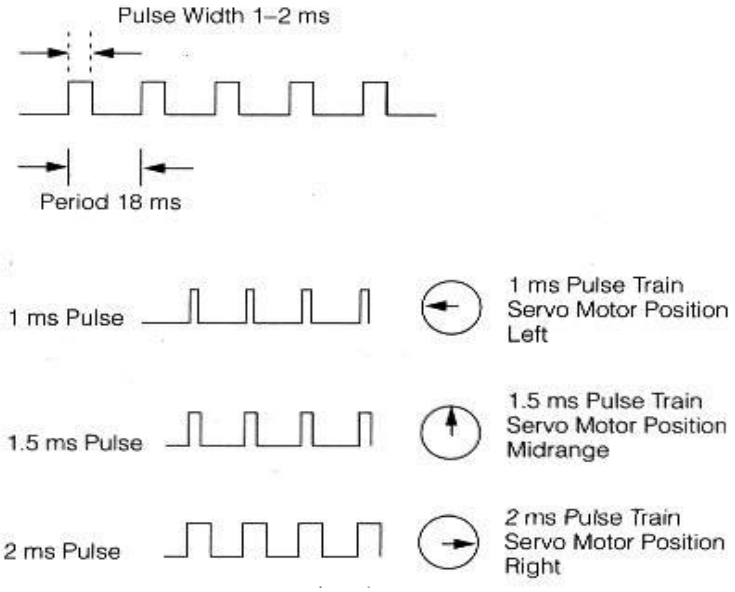


Illustration 1: Source: seattlerobotics.org

This pulse should be sent every 20 ms or so. The exact timing between pulses is not critical.

While we could easily write code to pulse the servo control the proper time (between 1 and 2 ms) every 20 ms, the Servo library used above takes care of that for us and can control multiple servos simultaneously.

NOTE: Servo range may vary; not all servos have a full 180 degree range.

Fail:

Unplug servo power line an plug in back in. Listen for servo to make a small move if power is connected properly.

Servo's can consume more power than available from the Arduino and from laptop USB port. Try connecting the external battery pack for additional servo power.

Check that the values printed to the Serial Monitor make see what's going on as the pot is moved.

Make sure the brown and red servo wires go to to GND and +5v respectively (call instructor if unsure).

Make sure the orange wire is connected to the correct PWM pin on the Arduino, especially if there is more than one servo connected. Change to servo control pin defined in the software if necessary.

If the electrical connections are suspect, try replacing the 3 pin headers with 3 jumper wires

CONTROL: Potentiometer Control of a Servo's Position Recipe (read a pot, move a servo)

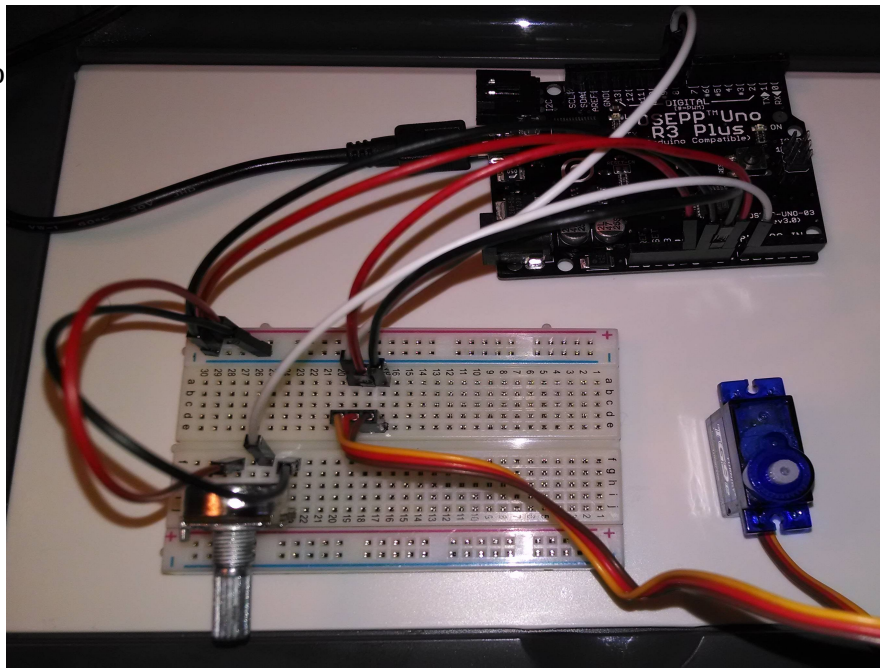
What:

In order to use a potentiometer to control a servo:

- Follow the the potentiometer recipe, connecting the pot output to the Arduino pin A0 (analog 0).
- Follow the Servo recipe, connecting the servo input wire to Pin 9 of the Arduino.
- Use the program below

How:

Here is a sample of how it could be wired up.



After testing the Pot and servo separately use the following code to test Servo control by a potentiometer.

Turning the pot left and right should cause a corresponding rotation in the servo.

The servo may move “faster” or “slower” than the pot depending on the map in the code below.

(We are using the values 30 to 150 for the degree settings, because some servo's have difficulties at the extremes of their movement.)

```
#include <Servo.h>  // servo library
// create a servo object
  Servo servo0;

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
  servo0.attach(9);  // servo is attached to pin 9
}

void loop() {
  int servo0Setting=90;  // for the servo position later

  int sensorValue0;      // Read Pot value

  sensorValue0 = analogRead(A0);
  Serial.print ("Pot 0 value=");
  Serial.println(sensorValue0);
```

```
  // Map the pot reading to the servo degree setting
  //    we'll use 30 to 150 degrees for the servo

  servo0Setting=map(sensorValue0,0,1023,30,150);
  Serial.print ("Servo 0 setting");
  Serial.println (servo0Setting);


  // Set Servo position
  servo0.write(servo0Setting);  // tell servo to go to the designated
position


}
```

Try This: You can change the direction of the servo to potentiometer mapping by swapping the pot input values in the map statement.
e.g. change this:

servo0Setting=map(sensorValue0, 0, 1023, 30, 150);

to this:

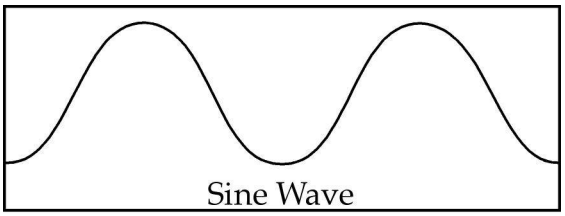
servo0Setting=map(sensorValue0, 1023, 0, 30, 150);

Of course different pins for the pot and servo may be used, but the test program will have to be modified to match.

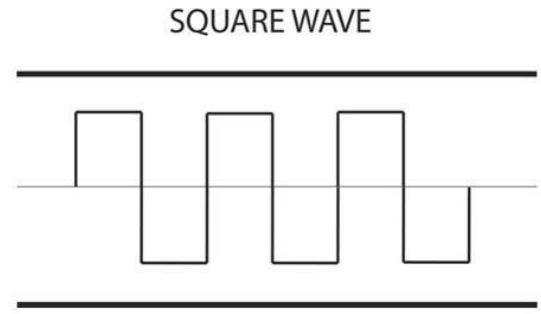
Fail:

OUTPUT: Sound Generation with Arduino (making tones, sounds and music with speaker)

What:



Sounds are generated by feeding a changing voltage into a speaker. How often the voltage changes per second is the frequency of the sound. A “pure” sound would be a signal that changes smoothly over time – like a sine wave.



Sine waves can be hard to generate digitally. But for making basic tones, a square waves are much easier to generate, require no real additional hardware and often works for most uses.

How:

Parts Needed:

- Speaker (The larger one in the kit. Not the 5v buzzer.)
- Red and black wires.

Assembly:

- Connect the black wire from the black side of the speaker connector to the ground hole on the Arduino.
- Connect the red wire from the red side of the speaker connector to DIGITAL pin 5.

> **NOTE:** The wires are a tight fit into the speaker connector. It may push the speaker wire out of the connector. So try to hold the speaker wire and connector while pushing in the other wires.

The Code:

```
int soundOut = 5;       // Sound output pin
int del = 2;            // This determines how long the signal stays
                        // HIGH or LOW in milliseconds

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

void loop()
{
  digitalWrite(soundOut, HIGH);
  delay(del);
  digitalWrite(soundOut, LOW);
  delay(del);
}
```

Pretty simple, right? The delay is in milliseconds. So each pass through the loop is a complete “cycle” and bit more than 4 milliseconds total. There is some overhead to executing the various statements so that is why it comes out more than 4, but only by a little and it's close enough for us. That makes the frequency about 250 Hertz. That's close to a “B” an octave below middle “C” on a piano. Here's a chart for reference:

So let's modify it a bit to change tones back and forth – kind of like a European siren.

```
int soundOut = 5; // Sound output pin
int del = 1;      // Delay or duration of HIGH and LOW pulses
                  // Longer delay times = lower frequency
                  // So shorter delay 1 = high tone, longer delay 2 = low tone

int count = 0;    // Counter to control how long to play a high or low tone

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

void loop()
{
  digitalWrite(soundOut, HIGH);
   delay(del);

  digitalWrite(soundOut, LOW);
   delay(del);

  // Since del is the square wave's pulse time (inverse of frequency)
  // and count is a total amount of time, add del to count so we
  // always wait the same amount of time
   count += del;

  // If the desired amount of time as been reached, reset the
  // count and toggle the frequency (1 to 2)
  if(count > 500)   // if past tone count (try 100, 25, 10, 4, 3, 2, 1 !)
  {
    count = 0;      // reset the duration counter
    if(del == 1)    // a == compares two things
    {
      del = 2;      // toggles the tone
    } else {
      del = 1;      // a single = sets a variable
    }
  }
}
```

Well, that's getting complicated! And if we want our Arduino to do anything else at the same time, it's going to get pretty nuts!

But remember the PWM info from the motors & servos? That's a square wave too! So lets try that.

```
int soundOut = 5;       // Sound output pin

void setup()
{
  pinMode(soundOut, OUTPUT);

  // Doing this here shows that we keep generating sound
  // Even after setup is complete and the loop can do
  // other things
  analogWrite(soundOut, 128);
}

void loop()
{
}
```

That's handy! See what happens if you change the analogWrite value.

But there's a limitation here. The PWM signal the Arduino generates is 490Hz. So you can't change the frequency.

So here's a better trick...

```
int soundOut = 5;       // Sound output pin

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

void loop()
{
  tone(soundOut, 440);    // Surprise! Arduino can play sounds directly!
}
```

Wow, now that's handy! It's similar to the PWM technique in that the sound keeps playing by itself (try moving the “tone” command to the setup instead of loop). In fact, internally it uses the same hardware. But instead of a single frequency that varies the pulse width, you tell it the frequency and it's always a 50-50 high/low division. You can also add a third value for milliseconds (1000ms = 1 second) so it will only play for a given length of time. Pretty slick! Here's a tone chart.

Note	1	2	3	4	5	6	7	8
B	62	123	247	494	988	1976	3951	
A sharp/B flat	58	117	233	466	932	1865	3729	
A	55	110	220	440	880	1760	3520	
G sharp/A flat	52	104	208	415	831	1661	3322	
G	49	98	196	392	784	1568	3136	
F sharp/G flat	46	92	185	370	740	1480	2960	
F	44	87	175	349	698	1397	2794	
E	41	82	165	330	659	1319	2637	
D sharp/E flat	39	78	156	311	622	1245	2489	4978
D	37	73	147	294	587	1175	2349	4698
C sharp/D flat	35	69	139	277	554	1109	2217	4434
C	33	65	131	262	523	1047	2093	4186

Try This: Write twinkle twinkle little star! (try tone duration of 400ms and delay between notes of 500ms) and the chart above (starting in key of C).

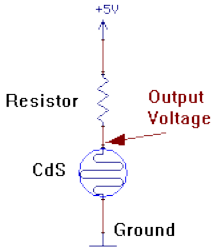
So why even bother with other sound techniques? All technology has pros and cons. Inthis case, there are some limitations with the “tone” command. For one thing, it can only play one tone at a time. So even if you have multiple speakers hooked up, you can't sent sound to all of them at once. Another problem is that since it uses some of the same hardware inside the Arduino, you can't use PWM on pins 3 or 11 at the same time you're generating tones. If these aren't a problem, and the sound quality is good enough, use tone.

If you're interested in these advanced techniques, check out “Advanced Arduino Sound Synthesis” in “Make:” magazine volume 35, page 80.

INPUT: Light Sensor (detecting light intensity with a CdS photo-sensor)

What:

A light sensing cell or photo-resistor or Cadmium Sulfie (CdS) cell turns light into a variable resistance. Usually dark = high resistance and high light = low resistance. In simple terms it is a bit like a potentiometer or volume knob for light.

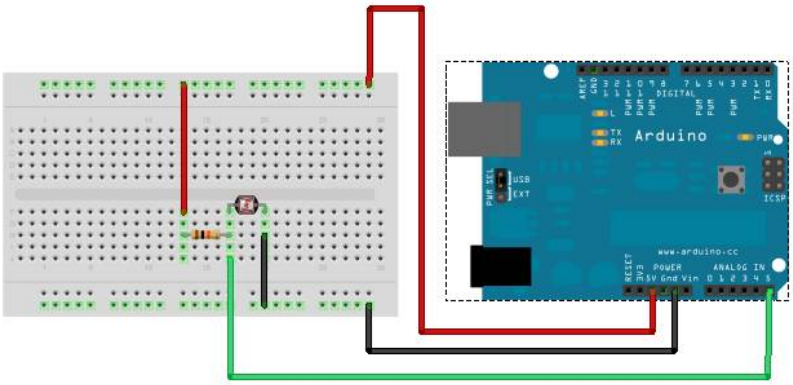


Just like a variable pot resistor, it can be used as an input to change programming values for time delays, LED light intensity, or even motor movement or sound!

How:

This circuit is pretty straigh forward. It is a light sensor and a 10k ohm resistor wired in series (between +5V and GND on the breadboard), and you take the voltage measurement from where the sensor and resisor meet on the breadboard.

This will give you a roughly 0.5v (value in the arduino with a light shining on it) to 3.5v (value in the arduino in ambeint light) range read in from the arduino.



Wiring diagram for the 10k resistor and light sensor.
Source: http://wiki.xinchejian.com/wiki/Introduction_to_Arduino

Here's the code:

```
// With a 1k resistor in series with this light sensor, you should
// get around 900 range in the dark, around 600 in ambient, and
// around 2-200 range with an LED shining on it.
// This is for a CdS photo resistor cell with approximate attributes
// of:    DARK ~ 100k      AMBIENT ~ 2k      LIT ~ 0.7k

int ls = A5;              //analog pin to which LS is connected
int ls_value = 0;         //variable to store LS values,
                          //we'll set this to zero first.

void setup()
{
  Serial.begin(9600);   //Start the serial monitor
  pinMode(ls, INPUT);   //Tell arduino that this pin is for input
}

void loop()
{
  ls_value = analogRead(ls);     //reads the light sensor values
  Serial.println(ls_value);      //prints the LS values to serial monitor
  delay(50);                     //slow it down a bit
}
```

Try This: Ever hear of a Thermin? It's a musical instrument you can play by holding your hand over it! You can make one by combining this light detector with the last program from the sound recipe. You'll need to add their setup int variable and setup sections to your own, and then use the tone line below in your loop:

```
tone(soundOut, ls_value, 10);  //You just made a light Theremin
```

Tip: For a smoother sound, take out your delay(50); line!

Fail:

There's not a lot ot go wrong with this setup, however commong problems may include:

- not having the power—to--resistor--to--lightsensor--to-GND hooked up correctly
- not tapping your A5 input reading off where the light sensor and resistor meet on the breadboard.
- Not having all your variables from the two programs both integrated into a the three unified int, setup and loop sections.

FutureRecipes to Add

Arduino Coding
- program sections
- reference commands

Light and external LEDs
- 20ma limits
- resistor
- pin modes
- dim light means port wasn't set to output
- pwm

Drive a motor / transistor
- 20ma limits
- biasing a transistor
- pin modes
- pwm

Resistors and Ohms law recipe
Using a multi-meter
Ultrasonic Distance Sensor
Infrared detectors
Light Resistive Sensors (CdS cells)
Passive Infrared detectors
bluetooth
soldering skills

far future
GPS
9 DOF GYRO
USB hids devices