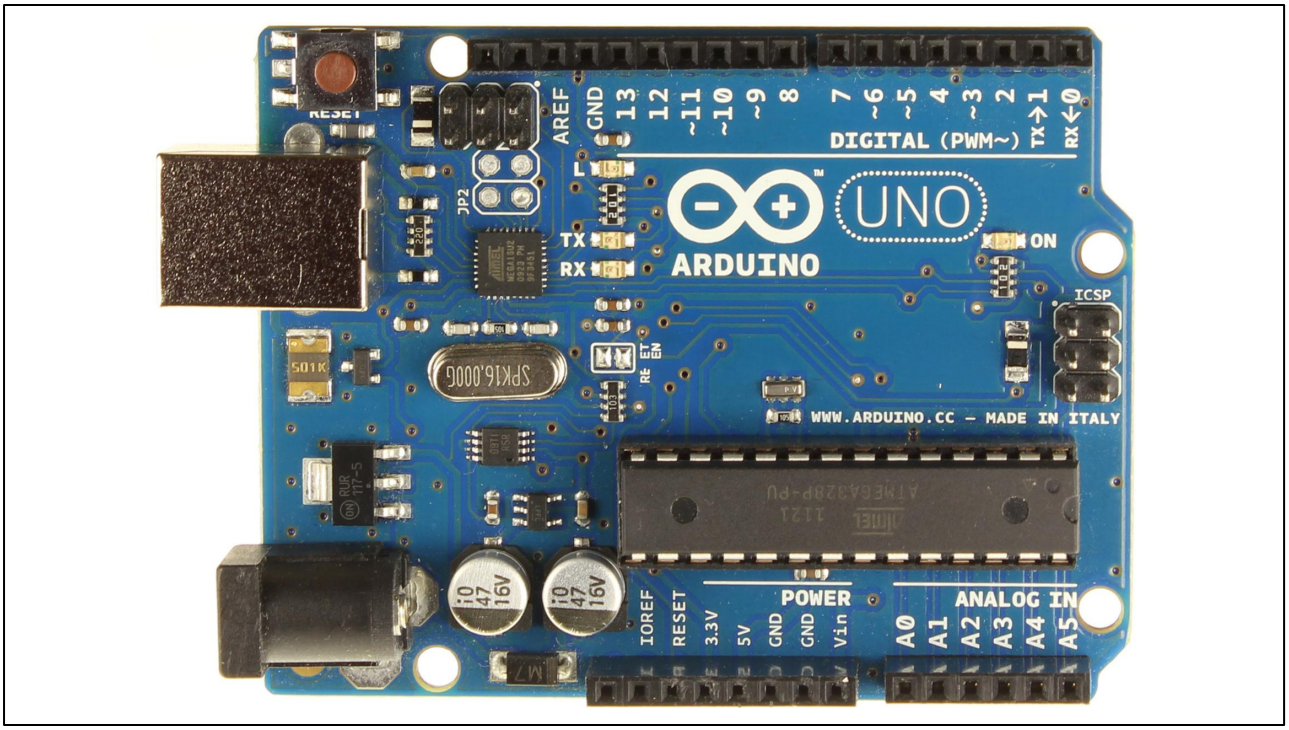
Analog I/O

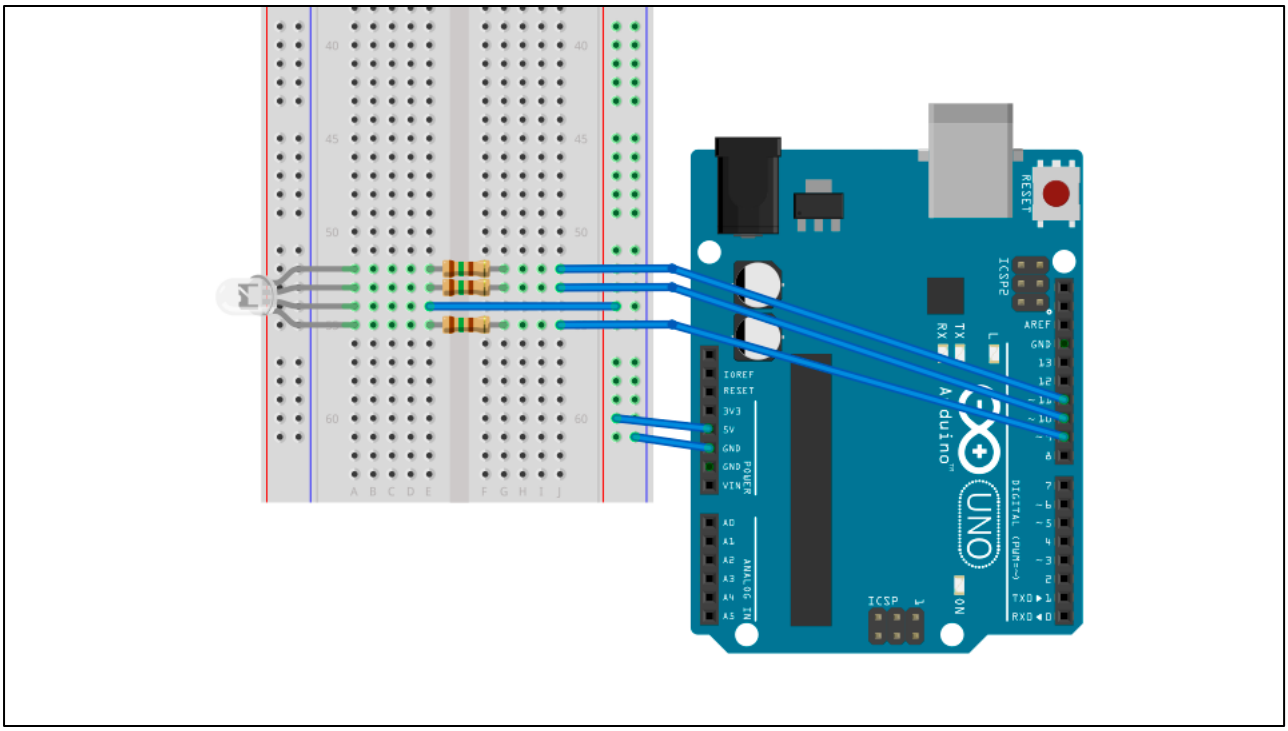
Let's talk about analog input and output! They are both pretty simple, actually. So far you have used the functions *digitalWrite()* and *digitalRead()*. Well, there is also *analogWrite()* and *analogRead()* and they work about the same way! The big difference is just that instead of having two states -- 0 and 5v -- the pin can deal with a bunch of in-between states.



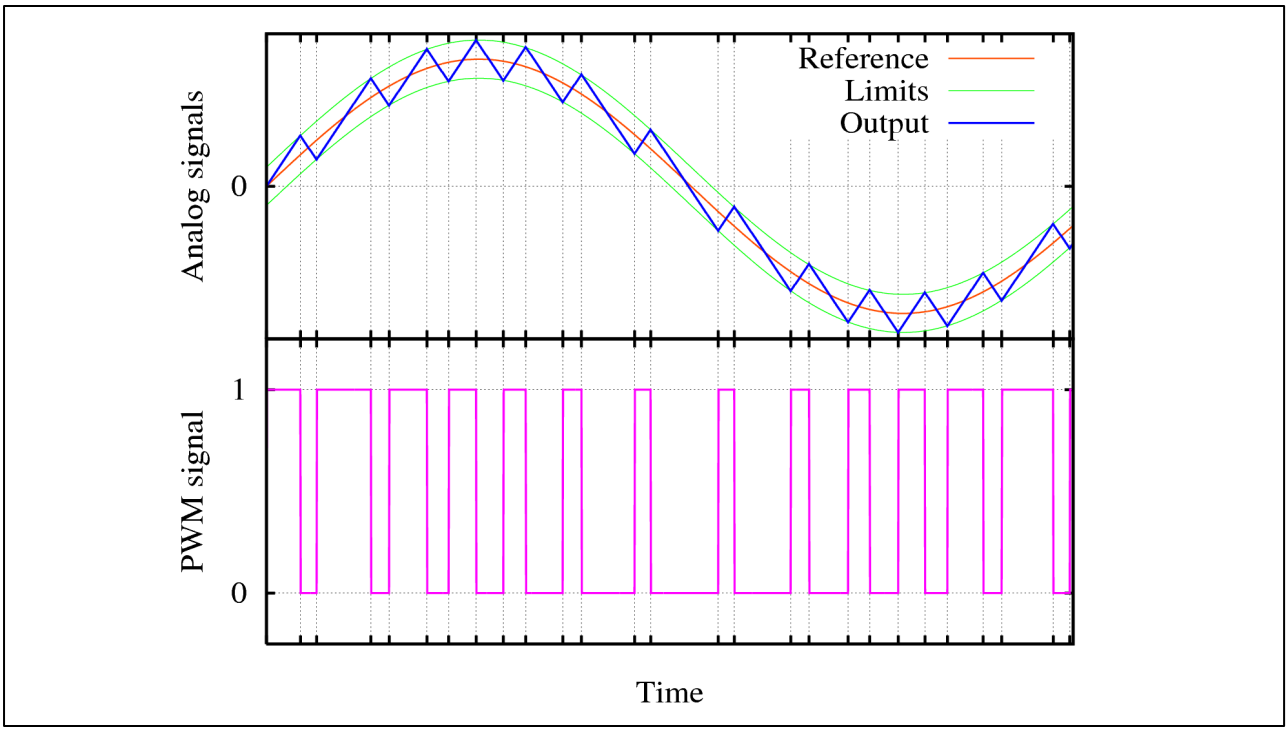
It's important to note that only some pins can handle analog input or output. A0-A5 can handle analog input -- that's because they're tied to the microcontroller's *ADC*, or *analog/digital converter*. Similarly, only some pins can handle native analog output. They're marked with a tilde, and include pins 3, 5, 6, 9, 10 and 11. If you need more, the SoftPWM library can add this output functionality to other pins, but it has some downsides. Use the aforementioned pins first, if you can.

File | Examples | Analog | Fading

For this lesson we're going to be using example code pretty heavily. Take a look at this example sketch and we'll walk through it. Change the ledPin variable to correspond to one of the pins that is connected to your LED.



As a reminder, here’s what we roughly expect your circuit to look like at this point.



In truth, microcontrollers don't REALLY generate analog values. They fake them! This is done by using *pulse width modulation* or PWM. A digital signal is turned on and off so fast that the power in the system approximates a given level. How much of the time it's on is called the *duty cycle* and ranges from 0% (0v) to 100% (5v)

When this translates into a physical system, it can be a real analog value. The above sine wave could represent the vibration of a speaker cone -- the PWM pumps tiny bits of energy into it and lets it relax in a precise way that simulates the underlying wave. The cone doesn't move instantaneously, which smooths the signal into something closer to the actual sine wave. In other cases it's just our biology that smooths it out: your LED is actually flickering under that PWM signal, but your retina can't keep up and so approximates a brightness level based on how many photons are hitting it over time.

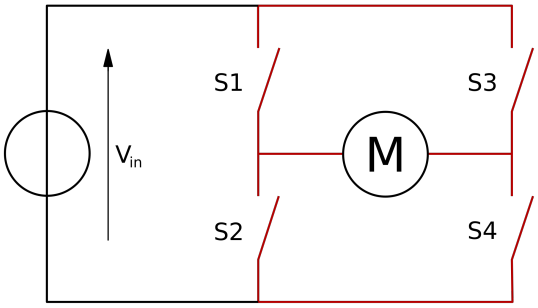
This is also how power supplies changed from being heavy transformer "wall warts" into tiny little cubes from Apple. Sometimes these systems produce lousy-quality signals -- note the choppiness -- and this can damage equipment. But there are ways of smoothing out signals, and these concerns are mostly just audiophiles being pains in the butt.

File | Examples | Servo | Sweep

So! We can fade LEDs. PWM also lets us manipulate servos! A servo is a special kind of motor that's designed to position its arm precisely within a limited range of motion. A zero percent duty cycle goes all the way to one end of that range, and 100% goes to the other.

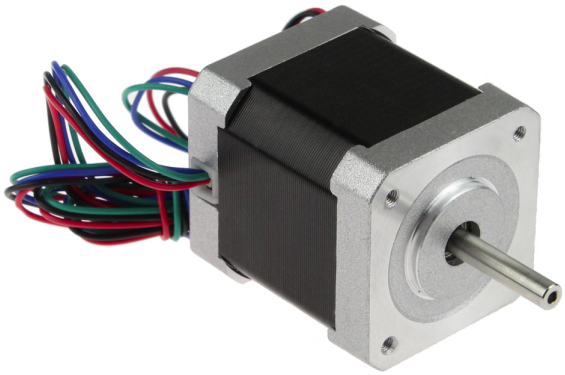
Servos have three wires: power, ground and the signal. Power is pretty much always red. Ground is usually black, but on these it's brown. The signal line is sometimes orange, like on these, but also sometimes white. Usually there's a datasheet that'll tell you. Anyway! Connect these things up, then load this program.

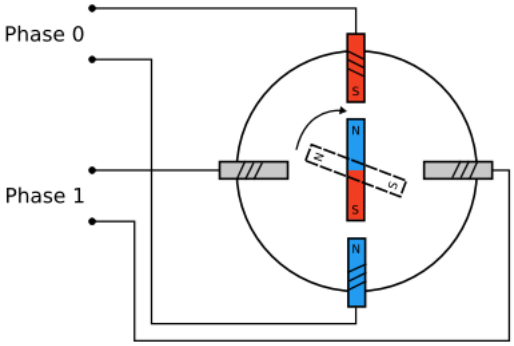
This is a more typical-looking servo. These are often used for steering mechanisms in RC cars, but there are many uses -- I'm sure you've heard the whine they make before. These microservos are about as small as they get, and cost between $2 and $4. You can get some really big servos -- how does 35 kg/s of torque sound? I hope it sounds like a lot, because it'll cost you $100. I have no idea how much it is, actually. Anyway you can good servos for ten bucks. Maybe make a cat door or a candy dispenser?



Here's a simple DC motor. These guys run when you give them power! One thing: you never really want to source power for a motor from a microcontroller. That's because motors use electromagnets, which means they represent an *inductive load*. As the electromagnet powers up, it generates an electric field. This field stores energy, drawing it in as it's established and releasing it as it collapses. This surge can be damaging to a microcontroller, so you generally want control of the inductive load to be somewhat isolated from your uC.

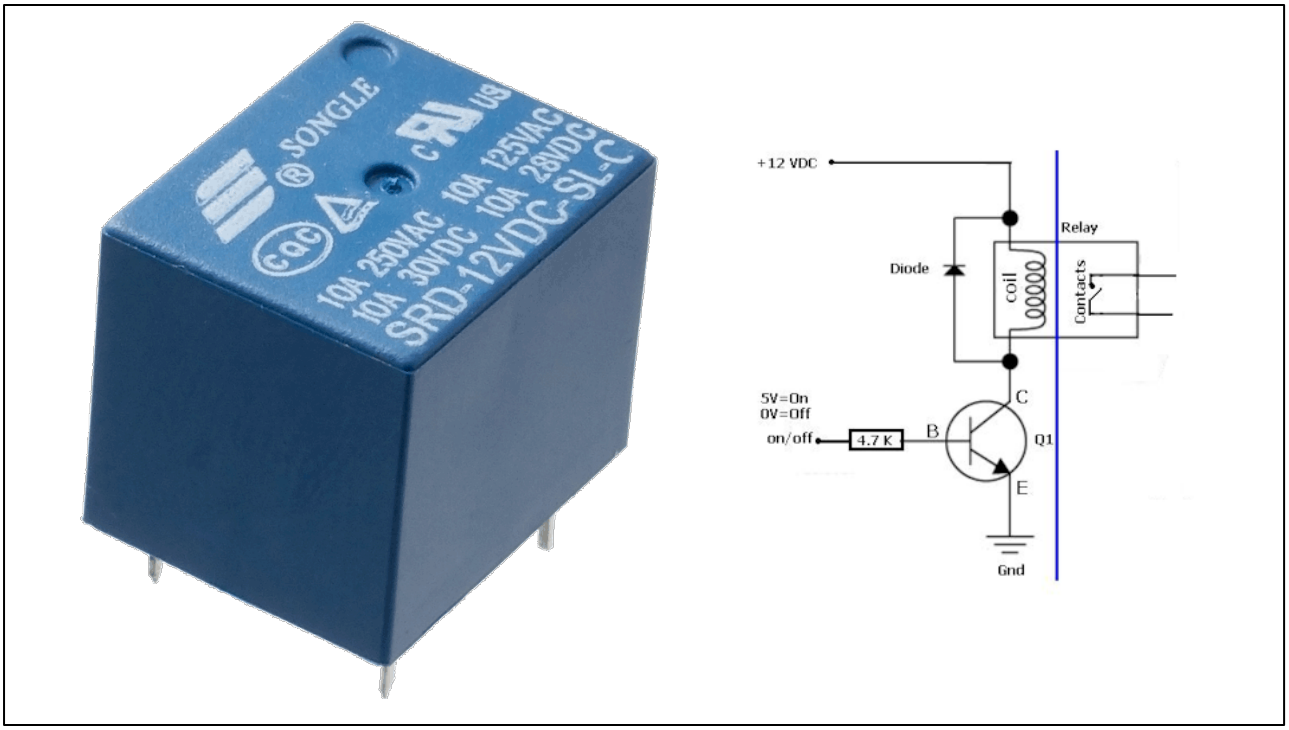
Here's one example of a circuit that helps with this: an H-Bridge. This is a very common circuit for control of a DC motor that both isolates it and allows it to be run forward or backward. When the motor (M) is running, either S1 and S4 are connected, or S3 and S2. One pair runs the current in one direction, the other in the opposite. The switches are actually transistors. But this is usually contained in a chip.





Here's another type of motor: a stepper motor. I haven't included a circuit, but the diagram illustrates the idea behind a stepper. The different phases are energized in a sequence that attracts the rotating part of the motor around in an arc, one step at a time. Each step is a set number of degrees, so this makes it very easy to control the positioning precisely. Some steppers have more phases than just two! But there are good Arduino libraries for tracking this stuff.

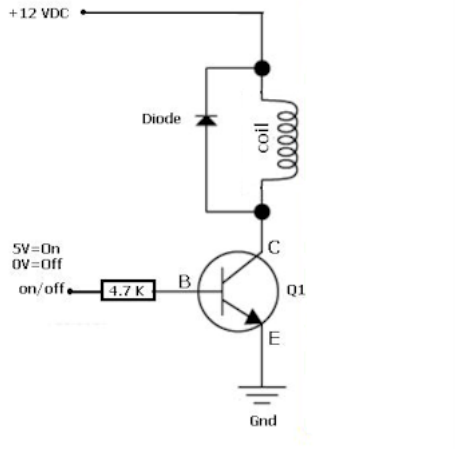
You can often get stepper motors out of disk drives, scanners or inkjet printers that are being thrown away (though not ones as beefy as the one pictured, which is more typical of a CAD/CAM or 3D printer setup).



This one isn't a motor, but it's still an inductive load. It's a relay! Here the magnetic field pulls closed a switch that's on a spring, allowing a much larger current to flow. You can use these to switch on wall-current appliances. The ratings on the case indicate how much energy it can handle (10 amps is a little less than a hair dryer, but still quite a lot). Relays come in many different shapes and sizes, but running off of 12 volts is fairly common, since that's the voltage level that automotive electrical systems run at. Fun fact: the clicking noise you hear from a turn signal in an older car is the sound of the contacts in the relay opening and closing.

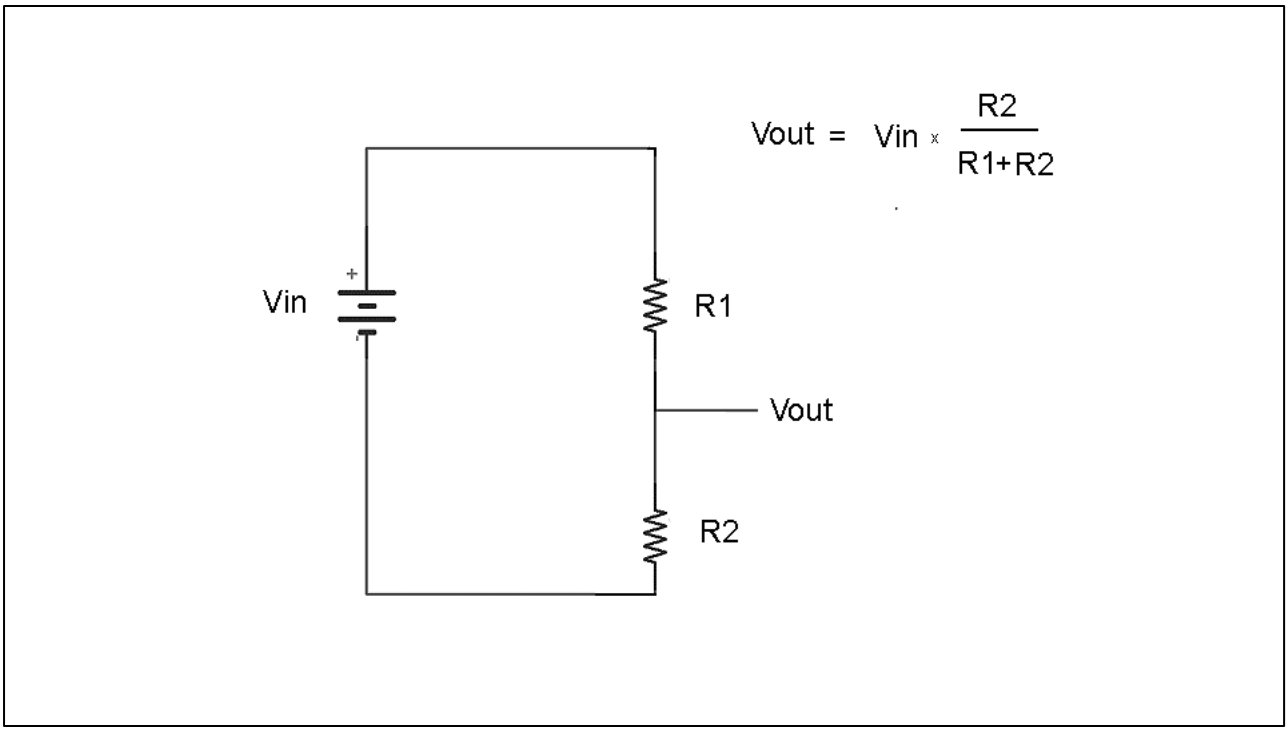
The circuit is worth looking at. Here there is a 12v input that powers the coil in the relay. It is only active when it's allowed to go to ground through a transistor -- Q1. The transistor lets this happen when a voltage is applied through a 4.7k resistor. This level is listed as 5v, so it could easily be coming from an Arduino.

The only other thing to note is the diode. When the coil's field collapses, current runs in the opposite direction of normal. It's possible for this to damage various components of the circuit. The diode prevents it from flowing that way.



Finally, here's a solenoid. The large circular part is just a coil. When it's energized, the metallic piston is pulled up into it. When its deenergized, it will pop back out, thanks to the spring -- this is what's known as a "captive" solenoid for that reason (the piston can't fall out). Solenoids are useful for creating quick pushing motions, like those of a car's electric door locks. Take care when using them, though: many are not designed to be kept on constantly, and will burn out if kept energized for long periods of time or too frequently. Their spec sheets usually tell you what's safe.

Note that the circuit for safely operating a solenoid is basically identical to that of a relay. That's because they're more or less the same device -- it's just that the thing being pulled by the magnetic field is slightly different.

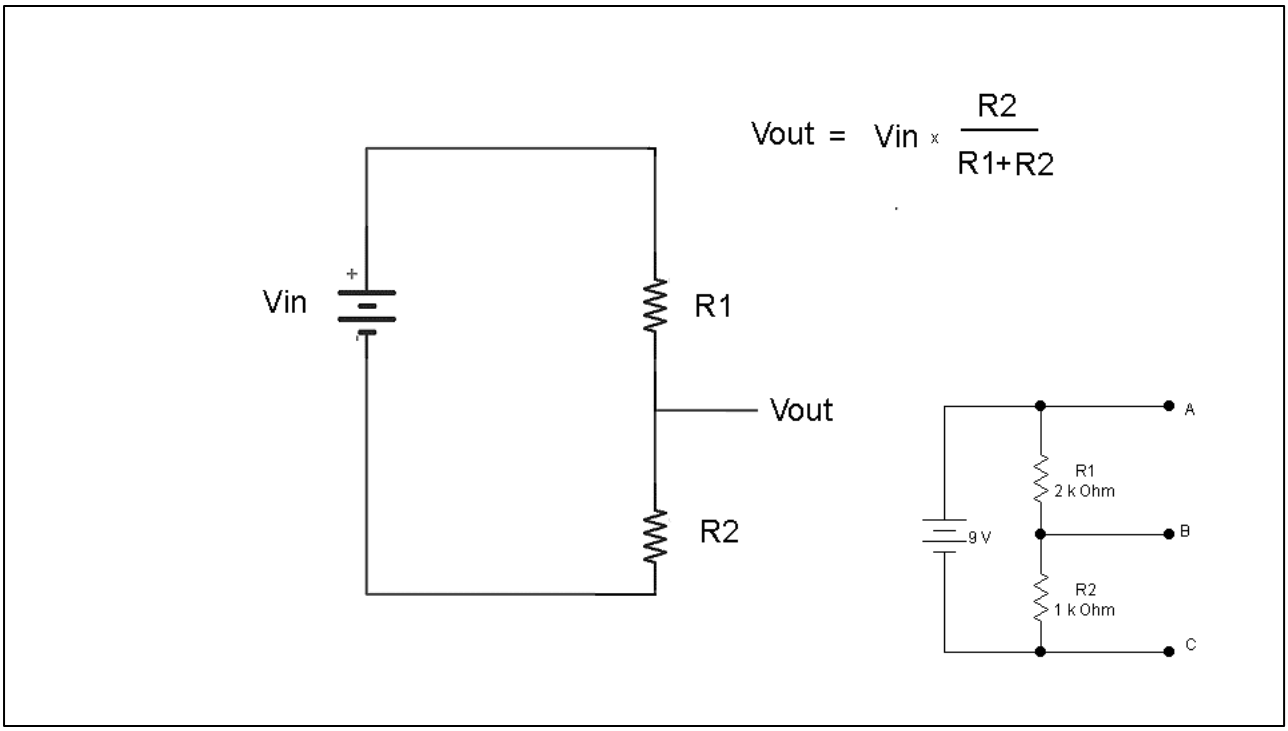


OK! That's it for our inductive devices. Now let's talk about analog input.

Analog input is possible on the Arduino's 6 analog input puts. Each one can read a voltage from 0-5v and convert it to a number from 0 to 1024 -- that's 10 bits of precision.

So: if we can make a voltage level vary in relation to something in the outside world, we can measure the outside world by measuring the voltage level. This is the idea behind many of the simplest kinds of sensors.

First we need to learn some basics about manipulating voltage levels with resistance. This is done through a *resistor network*. The math is actually quite simple. Consider the diagram above. How does the value of Vout vary based on the values of R1 and R2? The relationship is described by the equation.



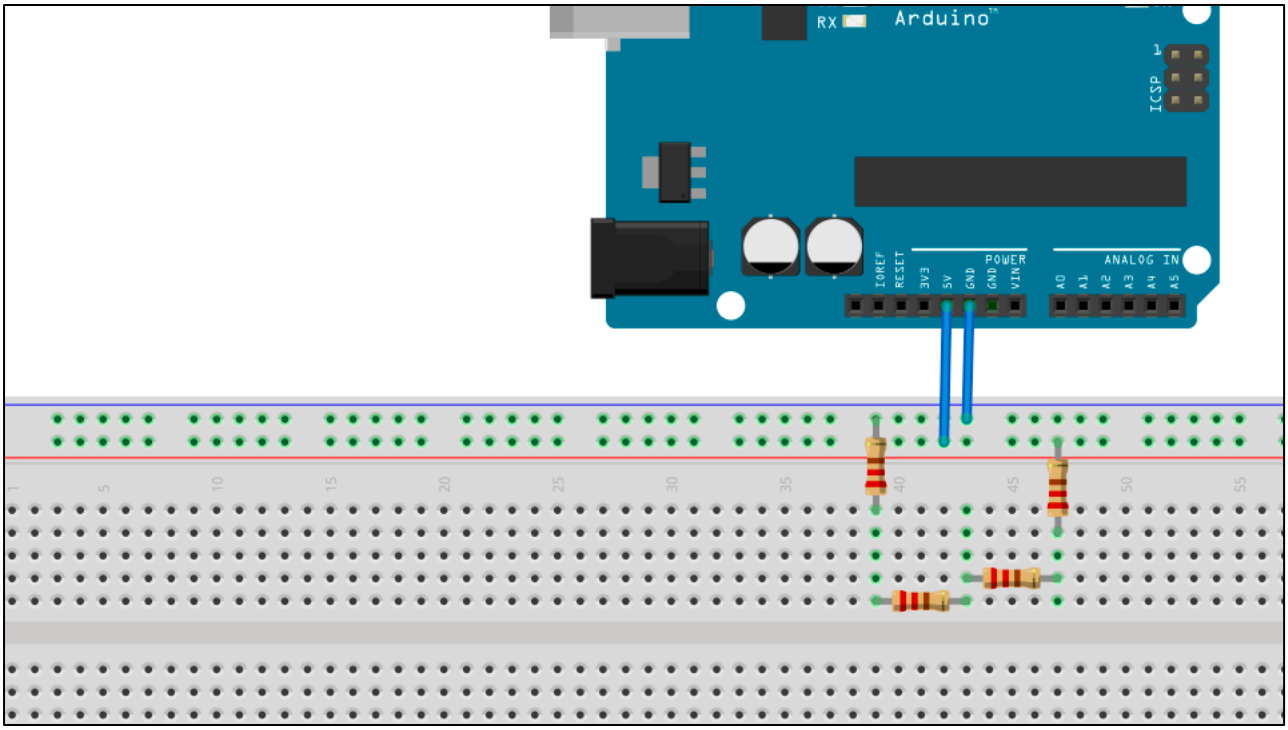
Try it with some actual values. What would be the voltage at point B if R1 is 2k ohms and R2 is 1k ohm and Vcc is 9 volts?

The answer is 3 volts. What is you swapped R1 and R2? What if the two were the same value? What if they were both only 100 ohm resistors instead of 1000 ohms?

(The answers: 6 volts, 4.5 volts, and still-4.5-volts-but-the-circuit's-normal-drain-would-be-greater).

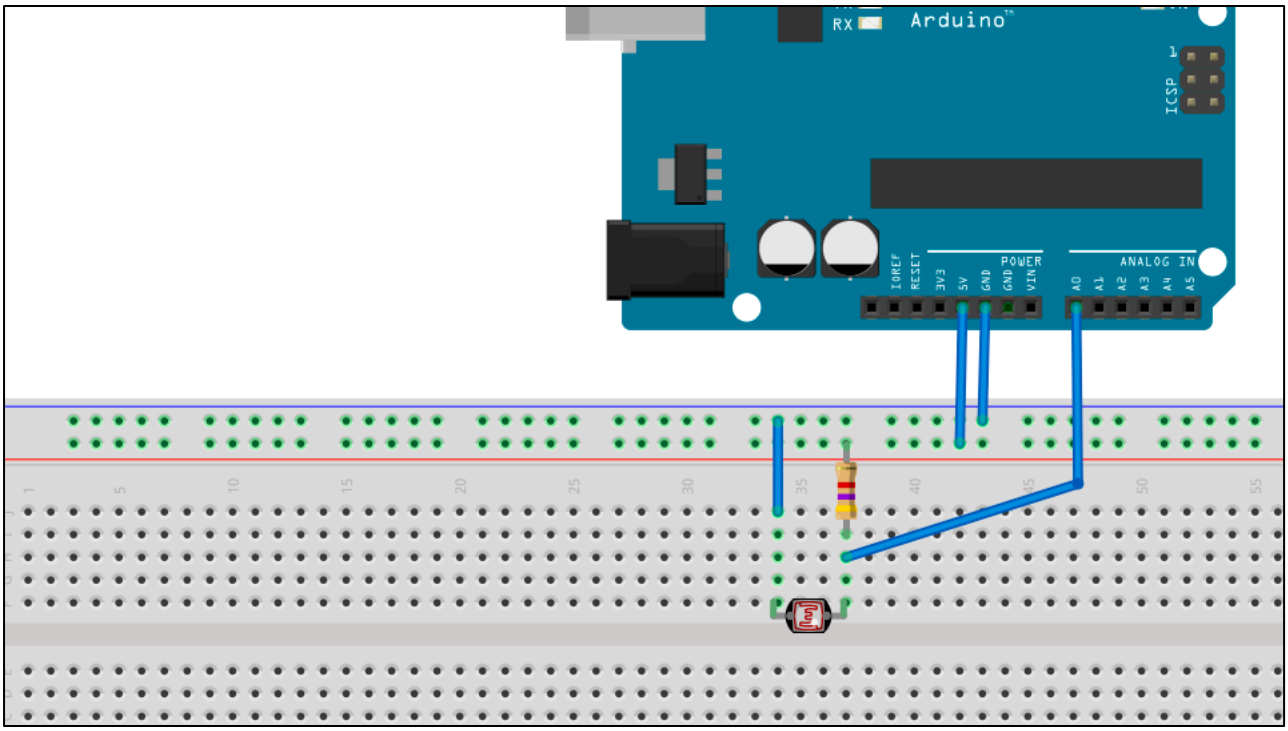
File | Examples | Analog | AnalogInOutSerial

OK! Load this on the ol' Arduino. Open Serial Monitor. Also, change the pins defined at the top. A0 can stay the same, but set the output pin to one of the PWM-capable pins that you have connected to an LED.



Create a connection of resistors in series like so. Use at least four (use more if you want). Then connect a wire to A0 and use it to probe each of the junctions between the resistors. Observe the varying values in the Serial Monitor (and the effect on the LED, if you have that wired up right).

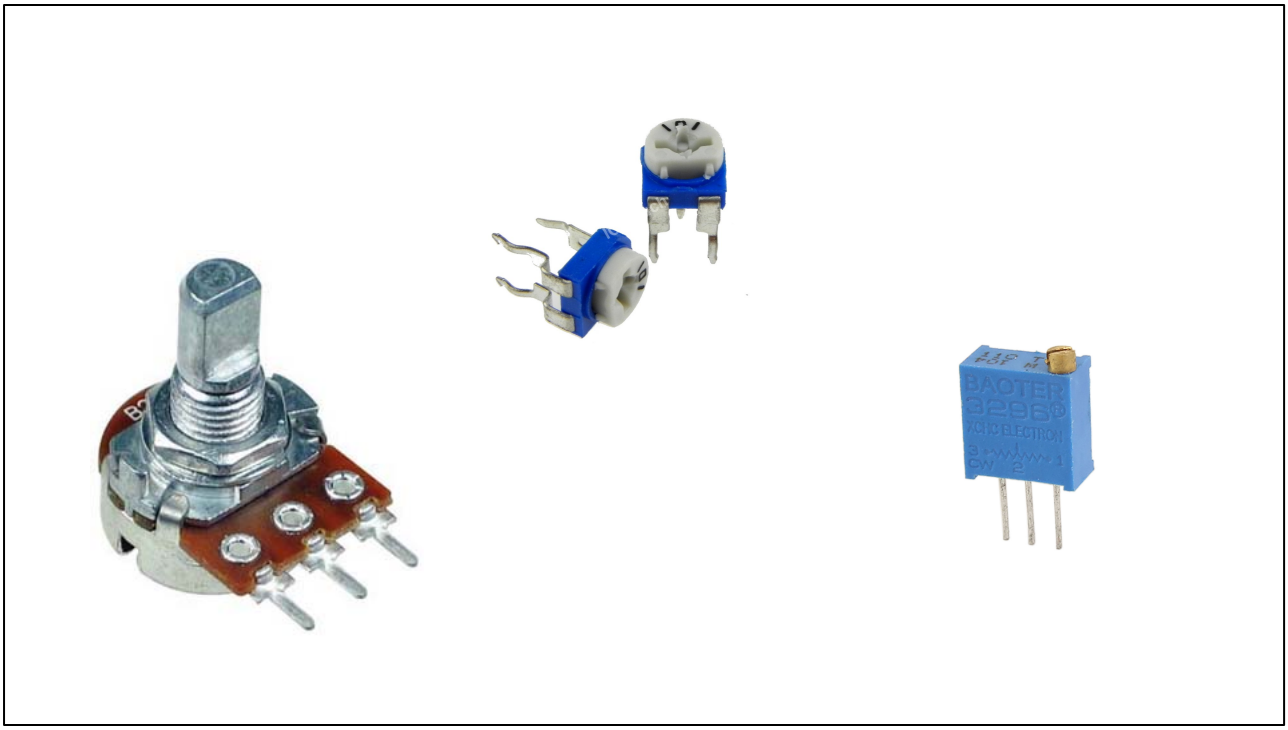
For extra fun, use the wire to probe the USB port (which is grounded), the 5v line, and your finger. Or let it dangle in the air as you bring your cell phone near it!



OK. Let's make a resistor network that actually measures something. This is a resistor network that's just like in slide 12/13. The only differences are:

- one of the resistors is a specific value, in this case 4.7k ohm
- the other resistor is a variable resistor. This one is a photoresistor that lowers its resistance as more light hits it. In my experience it runs from about 1k ohm to 10k ohm, depending on light levels. That's probably not exactly right, but it makes the 4.7k resistor an okay choice. You won't get the full dynamic range of the analog sensing input this way, but you'll probably get enough.

Connect the circuit as shown and observe the behavior in the Serial Monitor as you use your finger to change how much light hits the photoresistor. There are other simple sensors that change their resistance in response to heat, or bending, or magnetic fields.



What would happen if you changed the value of that 4.7k resistor? Have a look at the equation in slide 12/13 to try to figure out the answer.

In some cases, you'll want to be able to adjust the size of the 4.7k resistor, too. You can do that with a *potentiometer*, a device that lets you adjust resistance by turning a knob with your fingers or a screwdriver.

Anyway! I think you should get yourself a laser pointer -- a cat toy will do, but cheap ones can be had from eBay -- and point it at your photoresistor. And you should write a program that detects when the beam is broken and sets off an alarm. That alarm could be a lit-up LED, or it could be a serial message that prompts your computer to play a sound or send an email -- use your imagination!