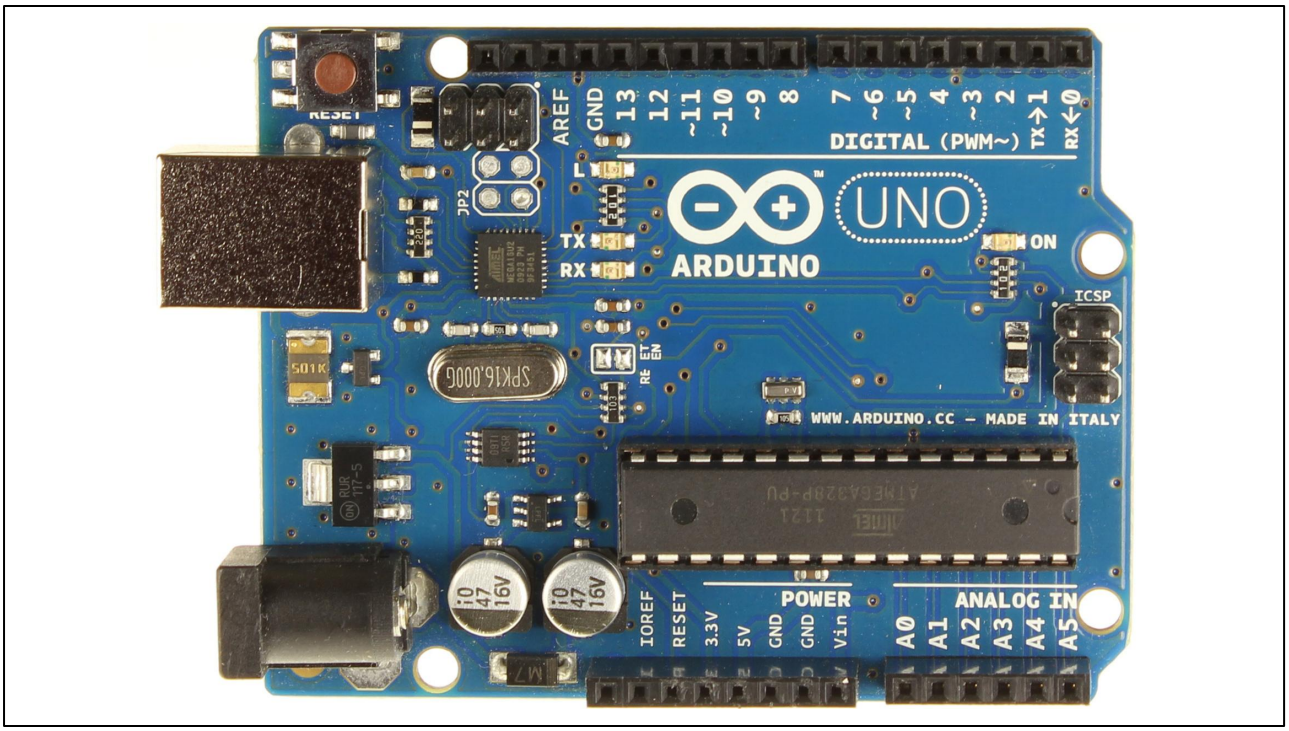
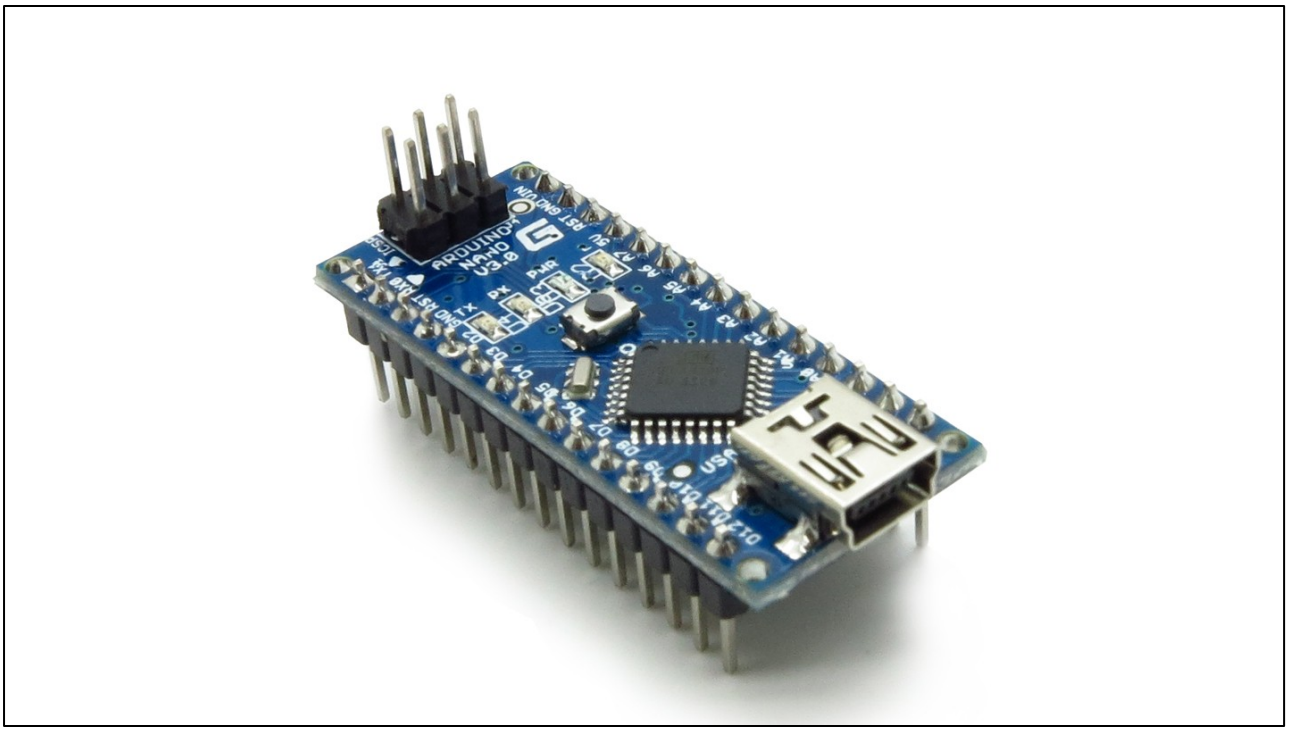
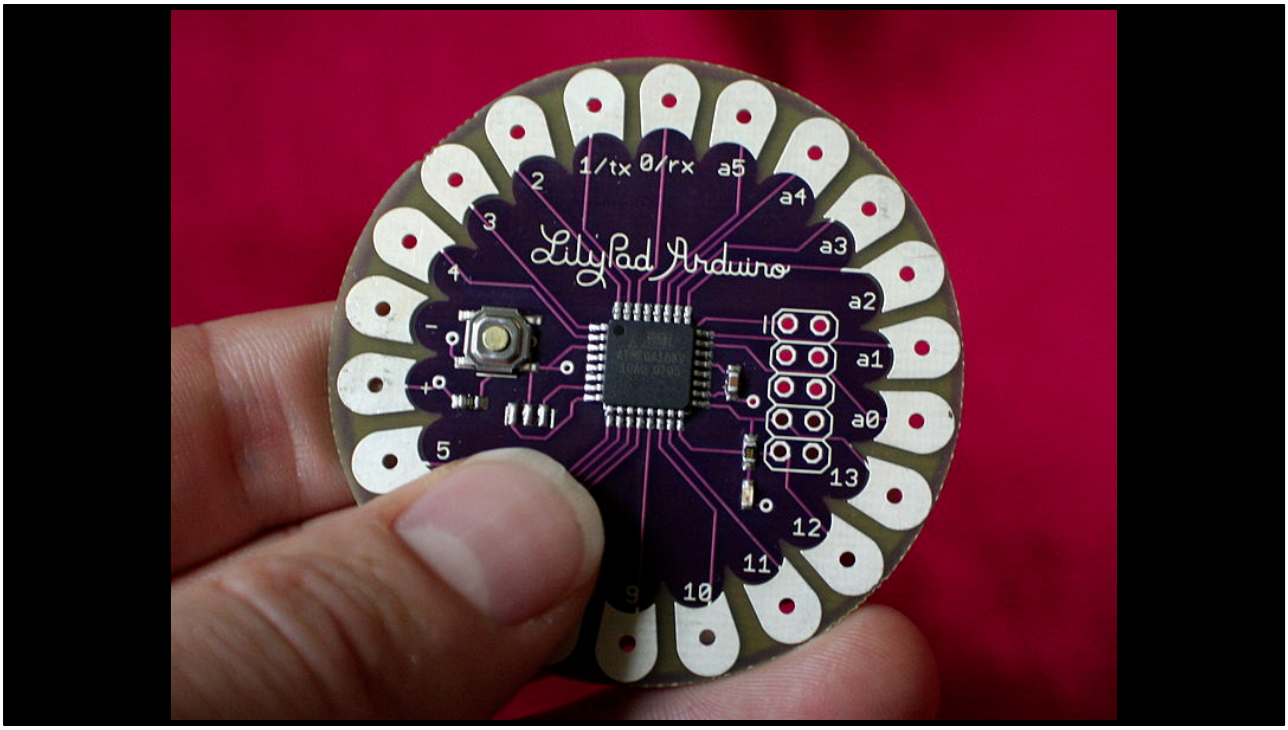


This is an Arduino! Important features: USB chip, clock crystal, voltage regulator (reduces erratic input current to a reliable 5V, throwing away extra as heat), protection diodes/resistors, LEDs, reset button.

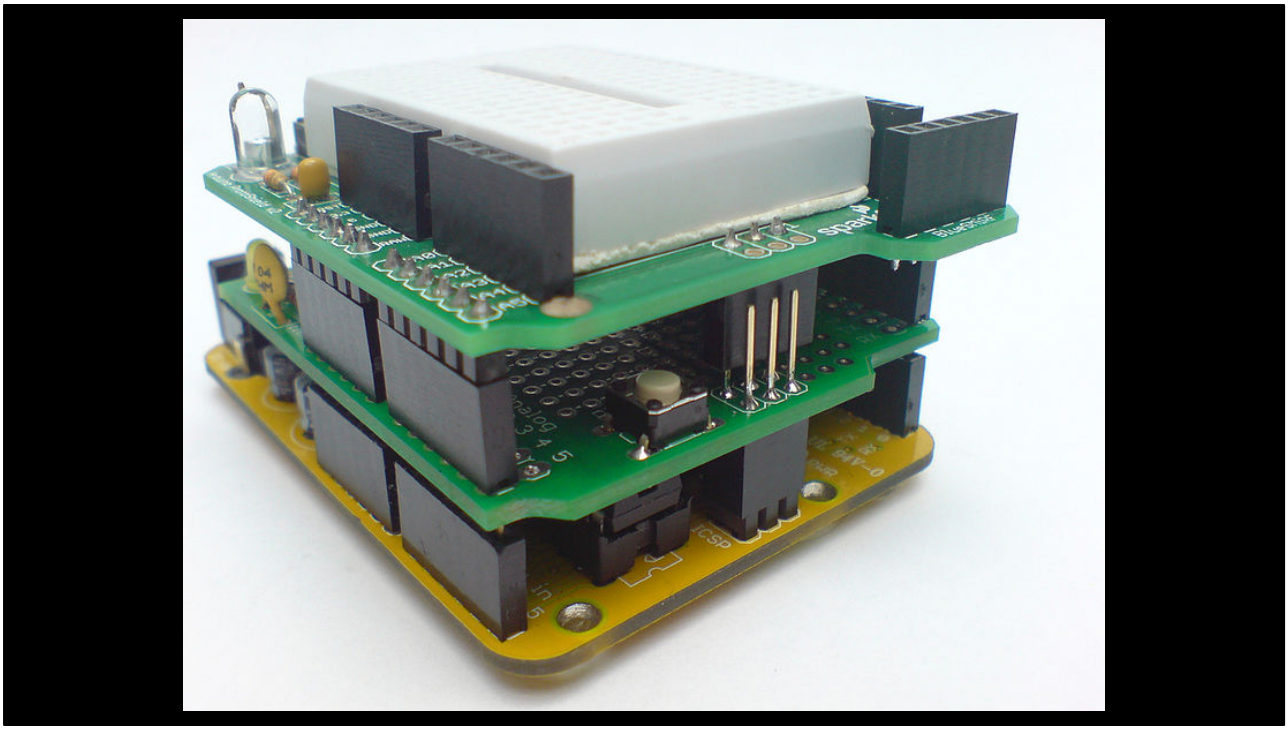


This is also an Arduino! it can be had for <$3 on ebay. Same stuff, but a less precise clock and the USB capability is designed to live in the programming cable (which costs about $15, but is only necessary during programming or communication with a computer).

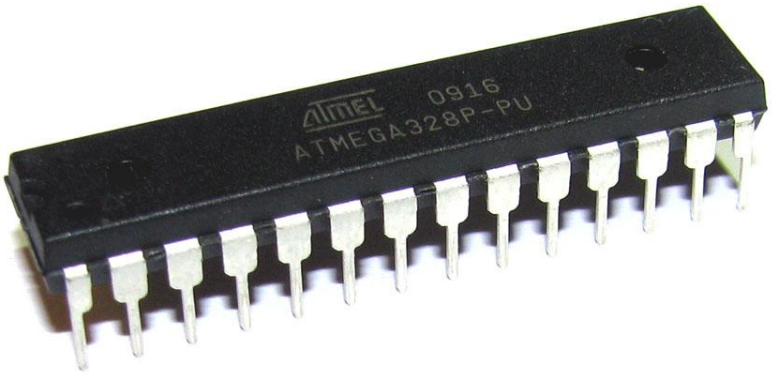


This is also an Arduino! The Lilypad is designed for wearable computing applications.

Note the different spacing of the pins on the integrated circuits across these Arduinos. The biggest one is in a form factor called “DIP” for “dual inline packaging”. This is the biggest and consequently the easiest for hobbyists to manipulate. It is what Arduino uses, but it is decreasingly common. In your childhood, your stereo and TV probably contained lots of DIP components. Now your Arduino is probably the only such thing.



These are shields, and one of the main reasons for using the original, big Arduino form factor from slide 1. The reliable position of the pins lets people design circuit boards for them, which can simplify the use of sensors, network interfaces and outputs by wiring up the support circuitry for you and providing software libraries.



http://www.atmel.com/Images/doc8161.pdf

This is an Atmel AVR microcontroller (sometimes abbreviated “uC” for µC, aka “micro” -C), and is the brains of the Arduino -- the rest of it is just there to support it, protect it, and help it speak to the outside world.

Microcontrollers are tiny programmable computers that are unaware of anything except the voltages being applied to their pins and their own internal sense of time (ie how many clock cycles have happened since they were turned on; not the time of day or year or anything like that). That’s it.

Microcontrollers run your microwave, parts of your car, parts of your laptop, and most other digital technology you use. They can have very low power consumption. And they are single-minded: a computer is doing lots of things at once, switching between each task. A microcontroller is not smart enough for that. If you need very precise or reliable function, they can be better than computers. You will never have one beachball!

What’s bad about them? They are low powered. They’re slow; most (including AVRs) can’t use decimals. Their networking capabilities are very primitive, and it’s not like they have an HDMI out. Simple things like data storage become a pain.

Microcontrollers are definitely among the most advanced integrated circuits you’ll encounter. The link on this slide goes to the datasheet of this microcontroller family. It’ s hundreds of pages long! A datasheet for a transistor might be two pages; one for a chip that controls a bunch of LEDs might be 20. One for a full on microprocessor like

the one in your laptop might be thousands. Still, AVRs can do a lot of things. The Arduino environment helps to simplify all of this detail and complexity in an easy-to-understand way.

```c
#include <avr/io.h>
#define F_CPU 1000000UL
#include <util/delay.h>

int main(void)
{
    DDRB |= (1 << 4);        /* Set PORTB bit 4 to output. */

    while (1) {
        PORTB &= ~(1 << 4);    /* LED on */
        _delay_ms(1000);
        PORTB |= (1 << 4);    /* LED off */
        _delay_ms(1000);
    }

    return 0;
}
```

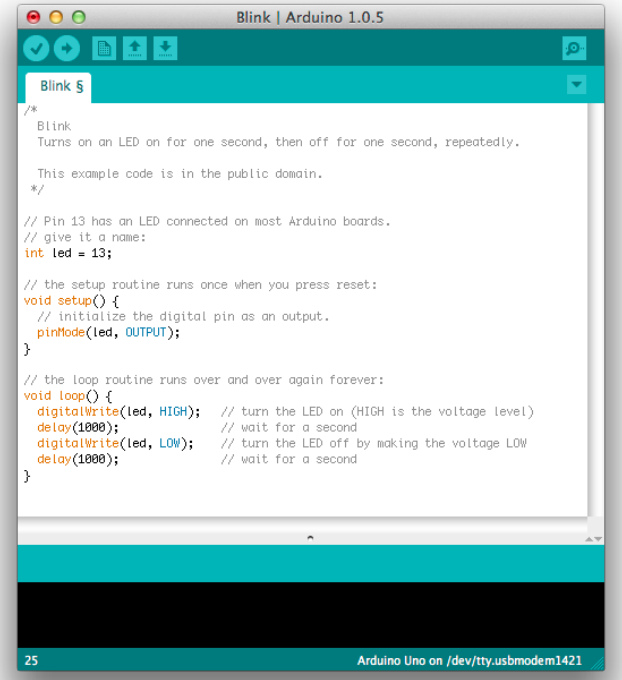
```c
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

https://gitorious.org/avr-gcc-examples/avr-gcc-examples/source/84eaa8519d3a124093934f4cba0197161bfc3af0:attiny13-blink/attiny13-blink.c

Arduino is a hardware platform that supports AVR microcontrollers. It is also a software platform that makes them easier to program.

On the left is some C code that programs an AVR to blink an LED attached to pin 13 on and off (with 1000 millisecond pauses between each state). The Arduino has a small LED built into it, connected to pin 13, for convenience (most other pins don't have this, with two exceptions that we'll discuss in future weeks). This is actually very readable code by microcontroller standards! But it involves lots of bitwise manipulation of obscure memory addresses. That's par for the course with uCs, but it' s a bit horrible.

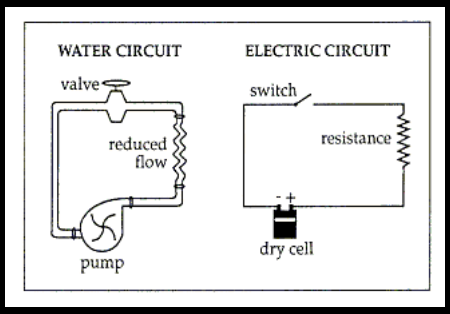
On the right is some Arduino code that does the exact same thing. Shorter and much more readable!



This is the Arduino IDE (integrated development environment). It's sort of awful, but it's what we've got.

Highlights: check & upload code buttons in the upper left. The address of the Arduino's serial port is in the lower right. Serial monitor (we'll use later).

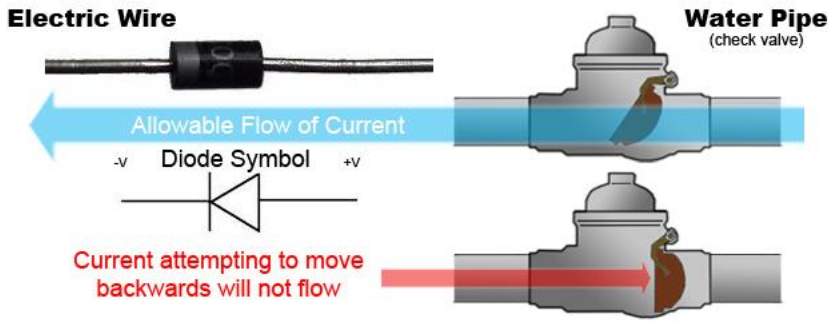
OK! Try to get your Arduino blinking. Go to File > Examples > Basics > Blink. Hit the upload button and confirm it works. Then try adjusting the delay values and uploading the revised program (in Arduino they're called "sketches").



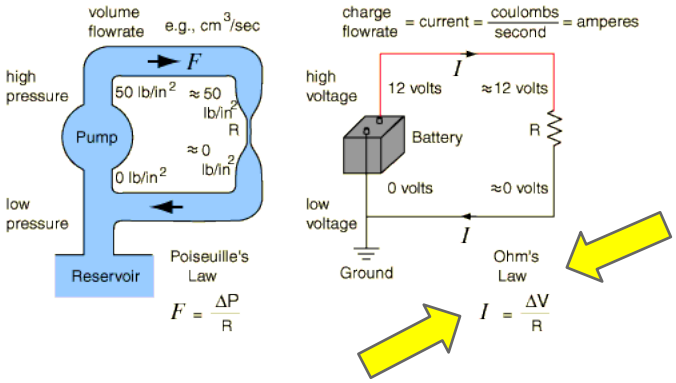
Alright. You know the basics of Arduino programming! The entire language can be found at http://arduino.cc/en/Reference/HomePage -- there isn't that much to it.

Now we have to learn something about electricity. This is where the frustration starts in the hardware hacking world. Reality works much differently (and worse) than software.

For now, I just want to plant a couple of ideas in your brain. One is the hydraulic analogy. This is a way of understanding the relationship between voltage, current and fundamental electronic elements. The analogy recasts them in terms of water. Here is a very simple example. The battery provides a push -- voltage, or pressure -- and that pressure differential can be used to do work. The on/off valve can arrest its flow by disconnecting the circuit. And the resistor/smaller section of pipe introduces a kind of friction.

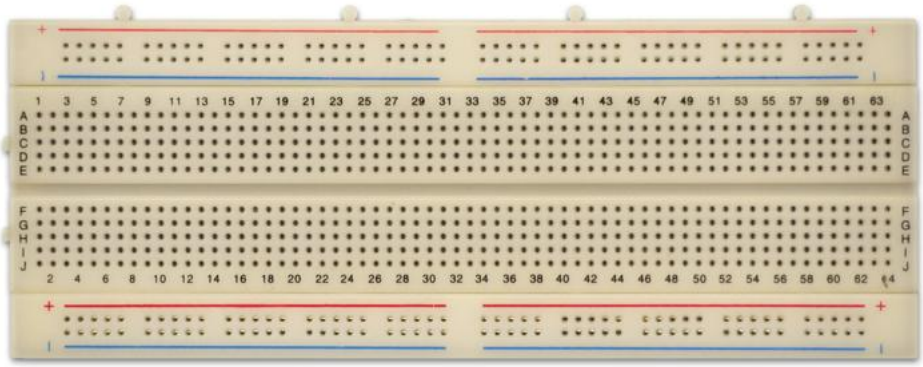


Here's a slightly more advanced analogy. A diode is an electrical component that only allows current to flow in one direction (a light emitting diode, or LED, is a special type of diode that also shaves off a few volts' worth of pressure and converts it to light energy). Here the analogy would be to a kind of water valve that only allows flow in one direction.

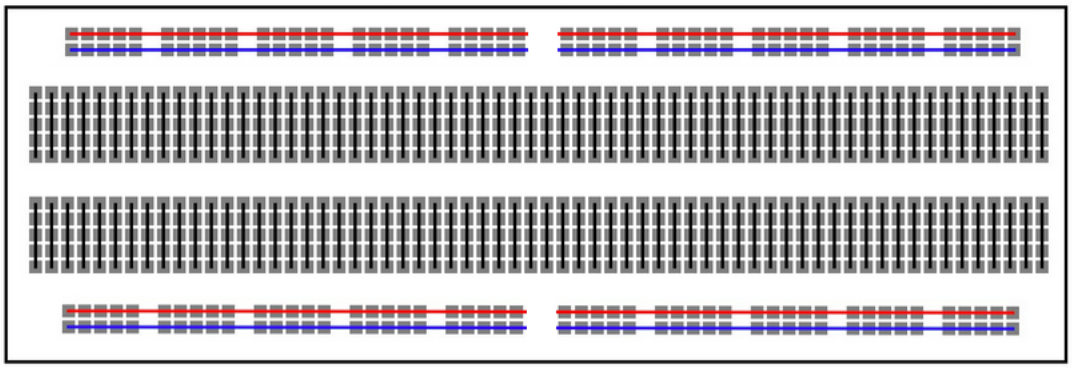


This diagram lays out more of the analogy. You don't need to understand it all. But do take note of Ohm's law: I=VR. This says that current equals the change in voltage times resistance. Note the similarity to Poiseuille's Law -- the hydraulic analogy really works!

Also note the difference in voltage on the two sides of the resistor, R, in the diagram on the right. It's 0 on the side connected to ground and 12 on the side connected to the battery's positive terminal. When thinking of a circuit, imagine ground like the vacuum of space -- it sucks things down to zero immediately. Imagine the positive-terminal (anode) side as if it had infinite capacity -- it will push the voltage up to twelve no matter what. Points of resistance between the two allow a trickle to pass from one to the other. For a moment, imagine there was another resistor, identical to R, between R and ground. What do you think the voltage between the two resistors would be?

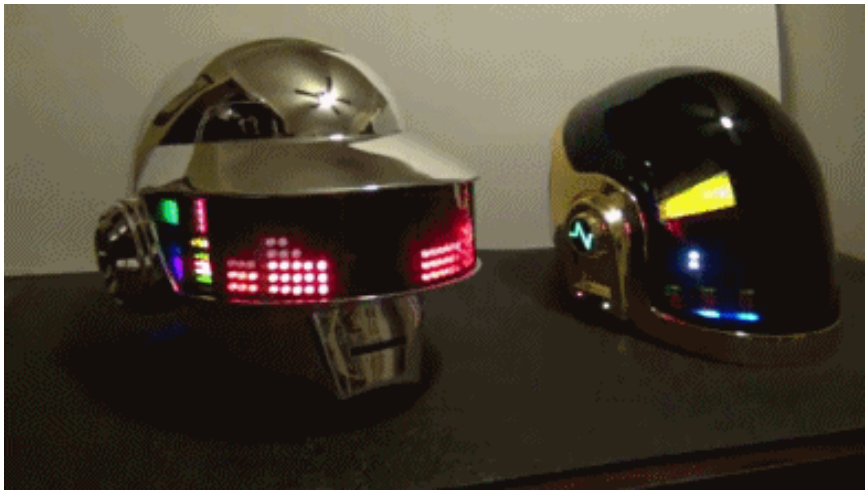


This is a breadboard. It’s a very handy device for prototyping circuits without soldering components together. Each hole has a spring-loaded grabby plate that will make good contact with wires inserted into it.

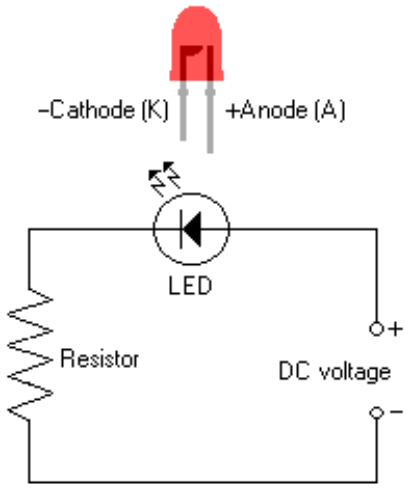


This shows the way that a breadboard's holes are connected together. The horizontal lines at the top and bottom are generally used for the supply voltage -- Vcc -- and for ground. By convention, Vcc is connected to red. The middle holes are where you get your work done. These are connected together vertically.

For our projects, we will be using the Arduino's power supply to run our circuits/act as Vcc. The Arduino runs at 5 volts. Many hobbyist devices now run at 3.3 volts -- the Arduino is kind of an exception. Be sure to check before connecting the Arduino to another component -- you could burn it out!



OK! Let’s make some things blink for real.



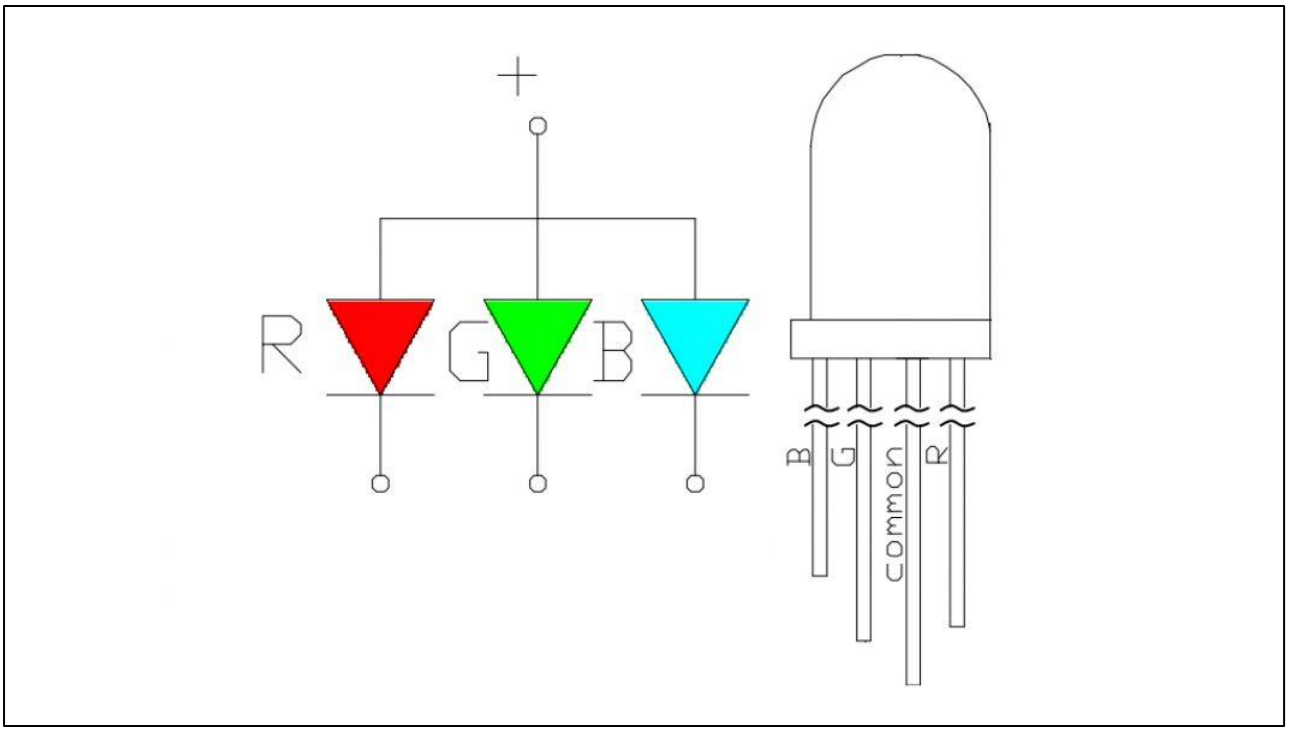
http://jumptuck.com/2011/11/07/using-leds-in-your-circuits/

http://led.linear1.org/1led.wiz

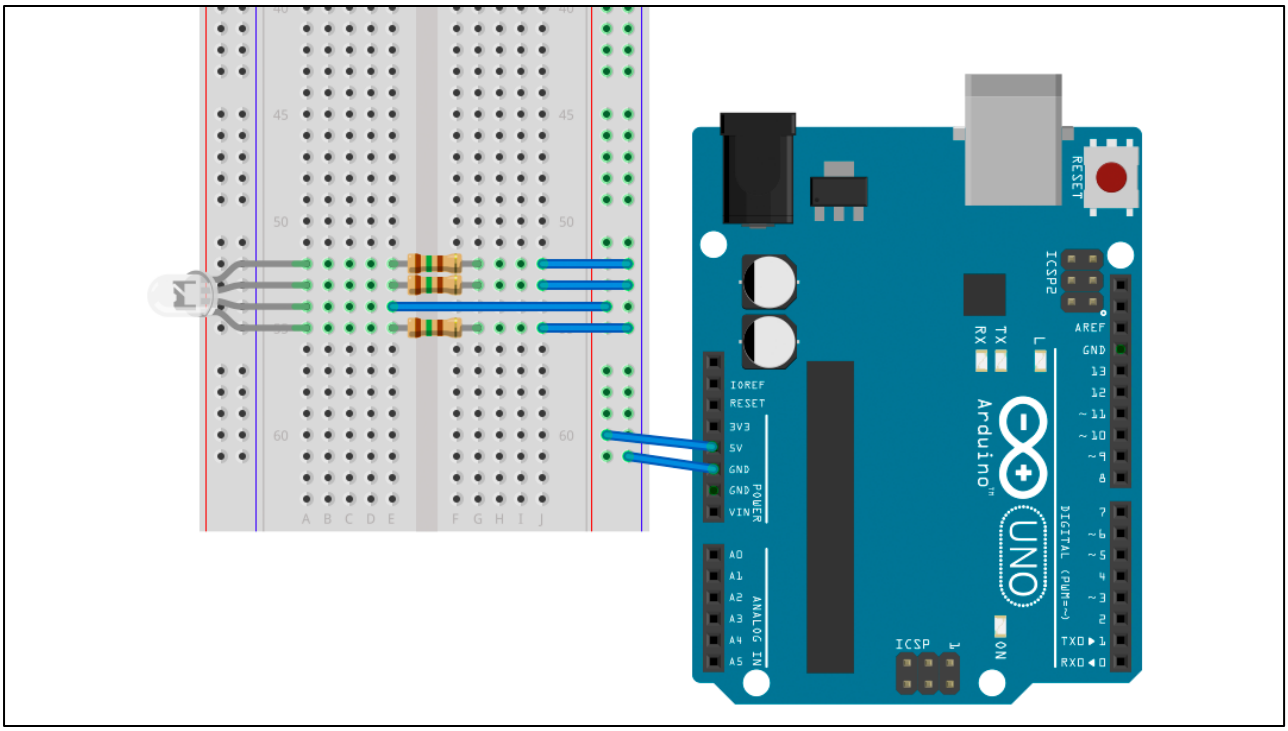
Here is a very simple LED circuit. In the hydraulic analogy, think of an LED like a delicate ornamental water wheel and the battery like a firehose. If you connect the two, the LED will work very briefly before destroying itself. You need some way to restrict the flow of power. That's where the resistor comes in. If you don't use one, your LED will always burn itself out immediately. It can be placed before or after the LED in the circuit -- it doesn't matter.

The precise value of the resistor can be calculated based on some of its electrical characteristics. I tend to just use the online calculator at the second link. We are just going to use a single, overly safe (aka large) resistor value for our LEDs regardless of their color, because it's simpler and because the human eye isn't great at distinguishing LED power levels anyway. The first link is also full of useful notes.

One last detail: LEDs have *polarity*, which means it matters which side is connected to positive and which is connected to negative (some components, like resistors, do not have polarity). By convention, the longer leg of an LED is the side that gets connected to positive. Don't worry if you put it in backward -- it won't hurt it unless the voltages are *very* high. But it will block current from flowing, because it's a diode.



In your kits you'll find some RGB LEDs. Use the clear ones for now -- I ordered the wrong kind of diffused ones. These LEDs are just like normal ones, except there are three of them in a single package. This lets you mix colors, which I think is kind of fun. To save space, RGB LEDs come as either "common anode" or "common cathode" meaning that they either share a Vcc or ground connection. Our clear LEDs are common anode. This means that the long leg will be connected to Vcc, and resistors will go between each color's leg and ground.



Let’s get the clear LED to light up using our breadboard. make these connections, tracing them through the breadboard. Imagine the charge starting at Vcc -- the 5 volt port. It goes into the red bank of the breadboard. From there we connect it to the long leg of the LED. It them goes into each color leg of the LED. Then it goes through the resistors. Those are each connected to the blue bank of the breadboard. Finally, that bank is connected to ground, completing the circuit. The LED should light up!

SINKING vs SOURCING

OK. Now let's combine the two. How would you rewire the breadboard to give the Arduino control of one of the colors in the RGB LED? You could start by connecting its resistor to pin 13 instead of directly to ground.

When we lit up pin 13 with the "blink" program, the LED illuminated every time the pin was set to HIGH. This makes sense: it's like sending energy to a light bulb, or like pressing a gas pedal to send fuel to a truck's engine. In electronics terms, we call this "sourcing current".

But remember, circuits work when electrons are flowing from areas of high charge to areas of low charge. It's all relative! Imagine I stood on a platform at the top of a Van de Graaf generator -- one of those things that makes your hair stand on end in a science museum. It could charge my body to tens of thousands of volts relative to ground. But if I turned on a flashlight in my hand, it would still work. That's because the batteries in it would be establishing a voltage differential the same way as always. Relative to the floor of the museum, the batteries terminals might be 10,003 volts and 10,000 volts. But that flow of 3 volts would work the same as always.

So! We can control the work of circuits not only by controlling access to Vcc, but by controlling access to ground. This is called "sinking current". It's a very common technique with microcontrollers, because for various reasons they can usually sink much more current than they can source (there are limits on both, though). In this case, if you connect one of the RGB LED's cathode legs to pin 13, you will notice that that sub-LED is illuminated at the times when the pin 13 built-in LED is off. The

Arduino is *sourcing* current to its built-in LED, and *sinking* current for the RGB LED. When it is *sourcing*, pin 13 is at 5 volts, the same as the LED's anode. There is no voltage differential, nowhere for charge to flow, and so no work is done -- no light appears!