Techshop EEE-201: Basic programming

Chapter 2

In this chapter, we are going to start building circuits and using our Arduino to control those circuits; we will be covering the following topics in this chapter:

- blinking an LED with `digitalWrite()`
- variable type `byte`
- using a normally open (NO) switch
- using a potentiometer to adjust the brightness of the LED

Table of Contents

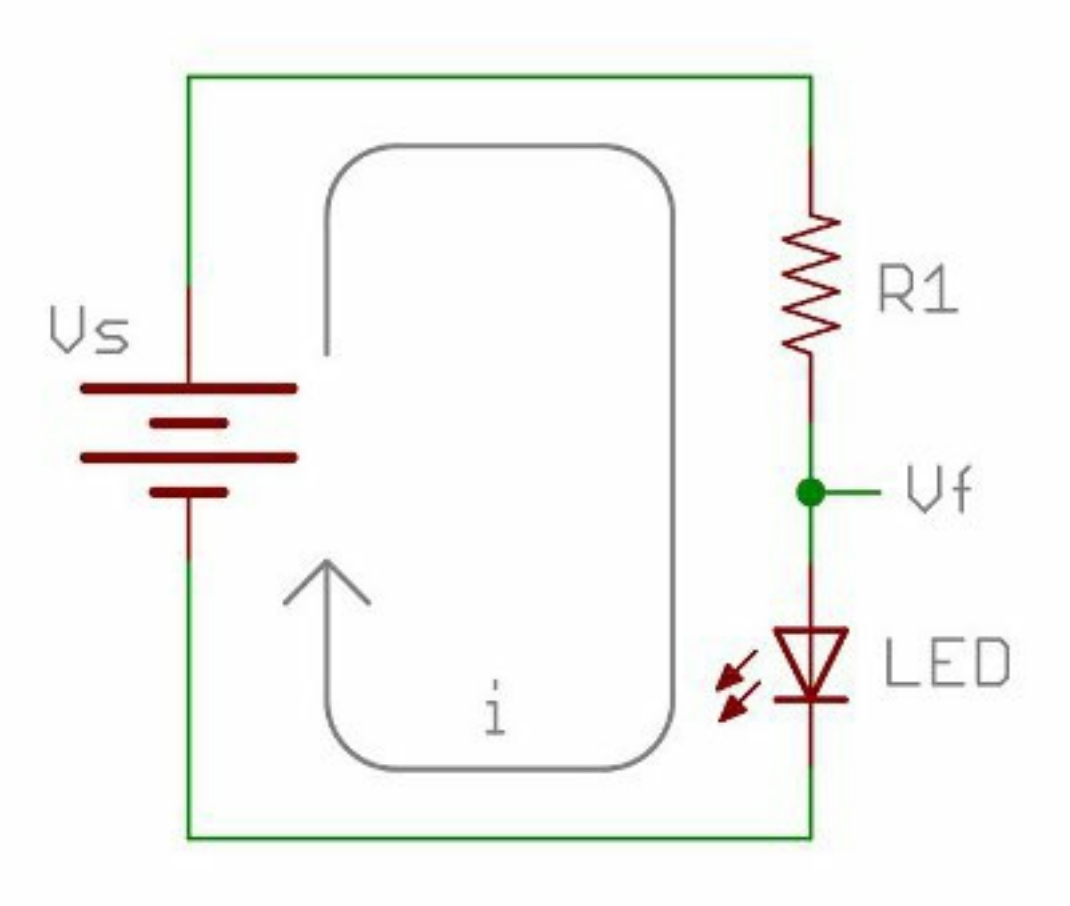
Introduction

This chapter assumes you are some knowledge of how to read circuit diagrams; if you have never seen one before, head over to the circuit diagram tutorial to read all about 'em.

Current limiting resistors

Limiting current into an LED is very important. An LED behaves very differently to a resistor in circuit. Resistors behave linearly according to Ohm's law: V = IR. For example, increase the voltage across a resistor, the current will increase proportionally, as long as the resistor's value stays the same. Simple enough. LEDs do not behave in this way. They behave as a diode with a characteristic I-V curve that is different than a resistor.

For example, there is a specification for diodes called the characteristic (or recommended) forward voltage (usually between 1.5-4V for LEDs). You must reach the characteristic forward voltage to turn 'on' the diode or LED, but as you exceed the characteristic forward voltage, the LED's resistance quickly drops off. Therefore, the LED will begin to draw a bunch of current and in some cases, burn out. A resistor is used in series with the LED to keep the current at a specific level called the characteristic (or recommended) forward current.



Using the circuit above, you will need to know three values in order to determine the current limiting resistor value.

`i` = LED forward current in Amps (found in the LED datasheet) `Vf` = LED forward voltage drop in Volts (found in the LED datasheet) `Vs` = supply voltage

Once you have obtained these three values, plug them into this equation to determine the current limiting resistor:

$$R = \frac{V_s - V_f}{i}$$

Also, keep in mind these two concepts when referring to the circuit above:

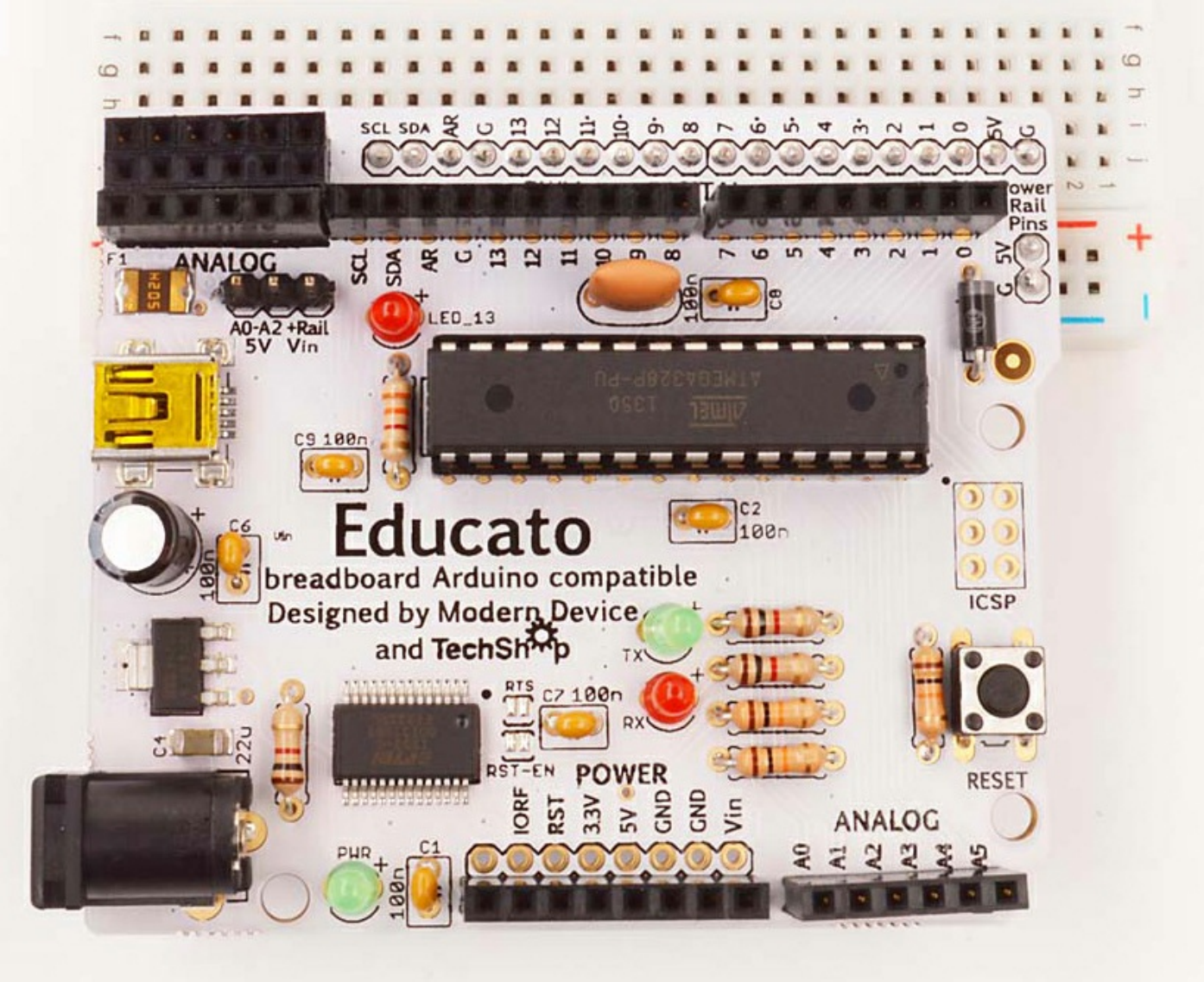
1. The current, `i`, coming out of the power source, through the resistor and LED, and back to ground is the same KCL
2. The voltage drop across the resistor, in addition to the forward

voltage drop of the LED equals the supply voltage KVL

You can find a handy resistor calculator online which will tell you exactly which resistor you should be using for your ciruit! When in doubt, always use a larger resistor than necessary because if you don't, you risk burning up the LED. In general, using a very large resistor will dim your LED; as you reduce the resistance, your LED will become brighter.

Header pins

Before we being using our Arduino, let's get oriented with the board:



On the board, we see multiple integraded circuits (ICs) such as resistors, capacitors, LEDs and so on; however most importantly we also find two

rows of black, female headers. These headers, grouped into three main sections (POWER, ANALOG and DIGITAL) will be the primary way we will interface with the arduino. They are made up of numerous pins that we will be able to control individually with our sketch.

POWER

This is where we can get our voltage from, both as 5V or 3.3V. We can also get access to *Vin* which will be the same voltage as your power supply (which can be up to 12V).

ANALOG

In the analog section we find 6 pins labeled *A0* to *A5* - these are analog inputs which will be described in Chapter 3

DIGITAL

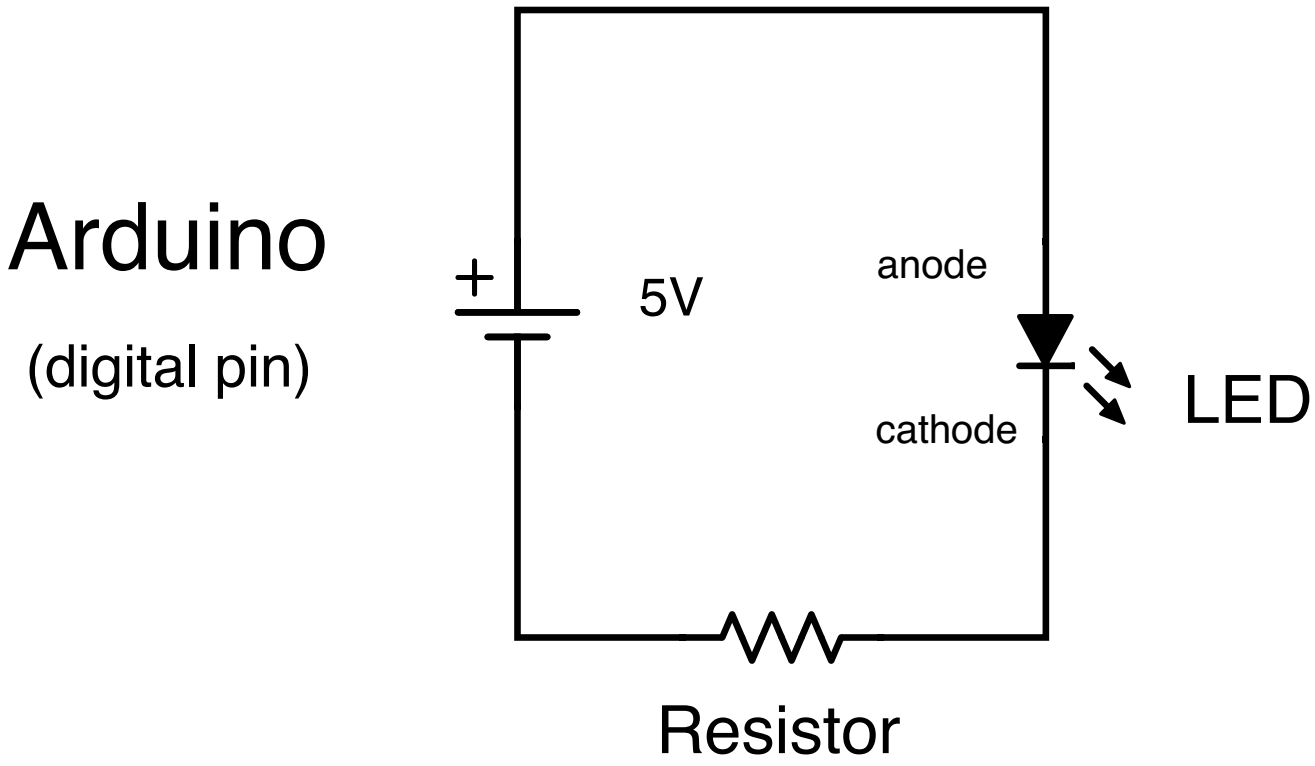
In the digital section, we have 14 pins labeled *0* to *13* (there are others but these pins are outside the scope of this class) - these are simple digital pins that we can turn either `HIGH` or `LOW` and will be covered in this chapter. Notice also that some of the pins have an asterisk by the number; this identifies these pins as supporting pulse width modulation (PWM), whic will be covered in Chapter 3

Part 1 - simple blink

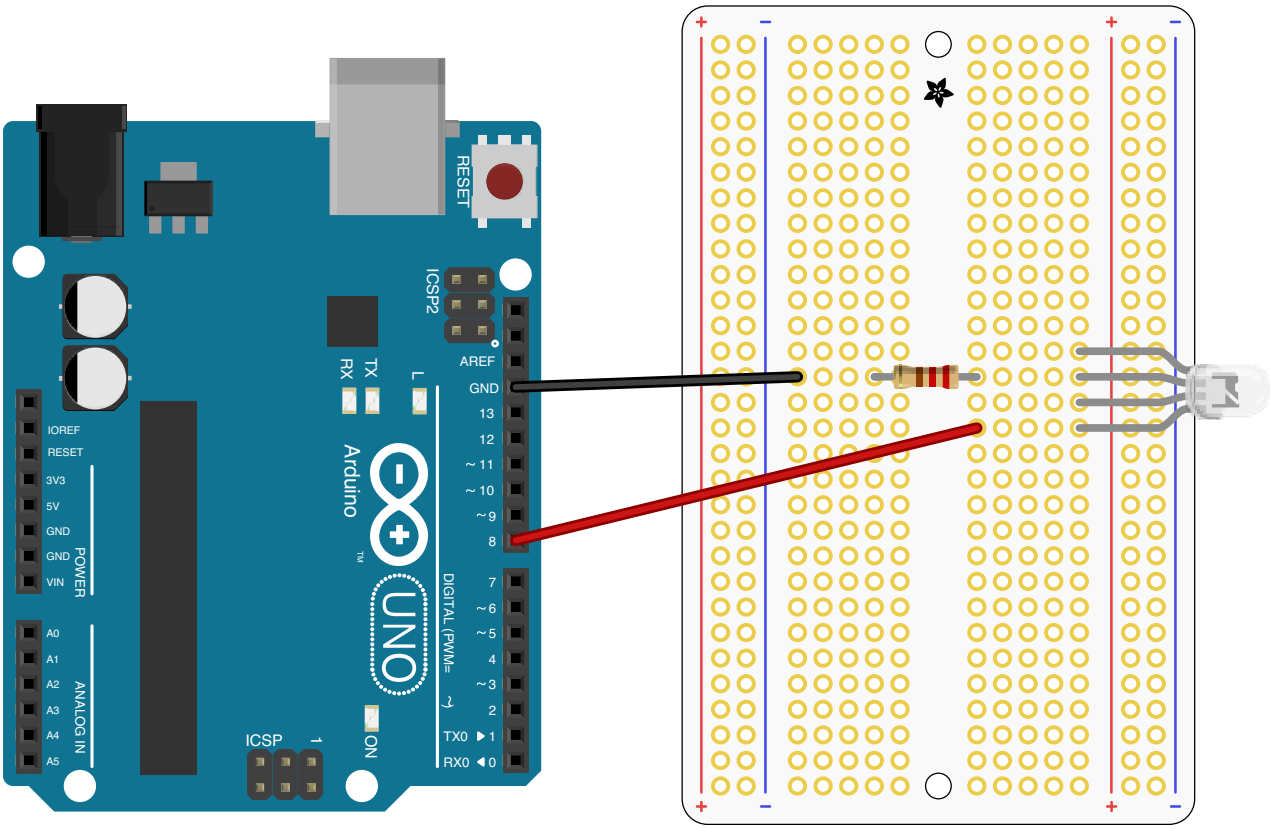
Circuit

We start by putting together a simple LED circuit which we will blink on/off with our Arduino. The circuit you'll put together on your breadboard can be found below; note that the sketch we'll be using

assumes you've hooked your LED up to digital pin **8**.



There isn't *one* correct way of putting the circuit together, but many different ways will work! To help you out, in the image below, we've shown you one way that you might put your circuit together.



You should be able to notice a few things with when comparing the image above to the diagram:

- We are using an RGB LED instead of a simple LED; you can think of these RGB LEDs as 3 LEDs put together in a single package. In the diagram, I've only hooked up the blue (B) part of the LED.
- In the image, current will flow in a counter-clockwise direction: starting from pin 8, going through the LED, then the resistor, and back to GND; in the circuit diagram, current is shown to flow in a clock-wise direction.
- We are taking advantage of the breadboard row #14 to connect the resistor to the black wire.

A few important things to remember:

- The resistor does not have polarity; that is, you can put it in any direction.
- The LED **does** have polarity: hook the anode up to pin 8 (5V) and the cathode to resistor (which then goes to GND).

Sketch

You'll be using the Blink sketch on the arduino. Note that the sketch assumes you've hooked your LED up to digital pin **8**.

Just like we saw previously in Chapter 1, our sketch has two parts to is: `setup()` and `loop()`; let's walk through the first part:

```
// the setup function runs once when you press reset
// or power the board
void setup() {
  // initialize digital pin 8 as an output.
  pinMode(8, OUTPUT);
```

```
}
```

In the `setup()` function, we need to tell the Arduino that we are going to use pin **8** and that we are going to use it as an `OUTPUT`; this is done through the `pinMode()` function. Setting it as an output means that the Arduino will be turning that pin `HIGH` (setting it to 5V) or `LOW` (setting it to 0V) based on the code in the sketch. Which is what we see in the `loop()` function:

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(8, HIGH);   // turn the LED on
  delay(1000);                       // wait for a second
  digitalWrite(8, LOW);    // turn the LED off
  delay(1000);                       // wait for a second
}
```

If you remember what we learned in Chapter 1, you should know that whatever is written inside the `loop()` function is, as the name implies, looped or repeated as long as the Arduino is on. In this case, we use the `digitalWrite()` to turn the pin 8 *on* or `HIGH` - what this means is that the voltage on this pin is turned on to 5V so that current can flow from the pin - this will turn your LED on! Next, the Arduino interprets the `delay()` function; this simply pauses the sketch for the duration defined in the function, in this case it's 1,000 miliseconds (or 1 second). The next two lines within the `loop()` function are exactly the same except that instead of writing `HIGH` to pin 8, we tell the Arduino to write `LOW` - think of this as turning off the current flow in the pin, setting the voltage to 0V - this will turn your LED off.

And that's it! This sketch will turn your LED on for 1 second, then turn it off for 1 second; this gets repeated forever!

Explore

- how would you have to change your sketch so that the LED blinks at 2 Hz (once every 0.5 seconds)?

Named variables

What happens if you'd want to change which pin controls your blinking LED? That's easy enough, just change all the 8s in your sketch to say, 9. That means we'd have to change 3 lines of code; that doesn't sound very practical. We can get around this by using variable names - you can think of these as a place holder for a specific piece of information. Let's start by an example, change your sketch to match the following and upload it:

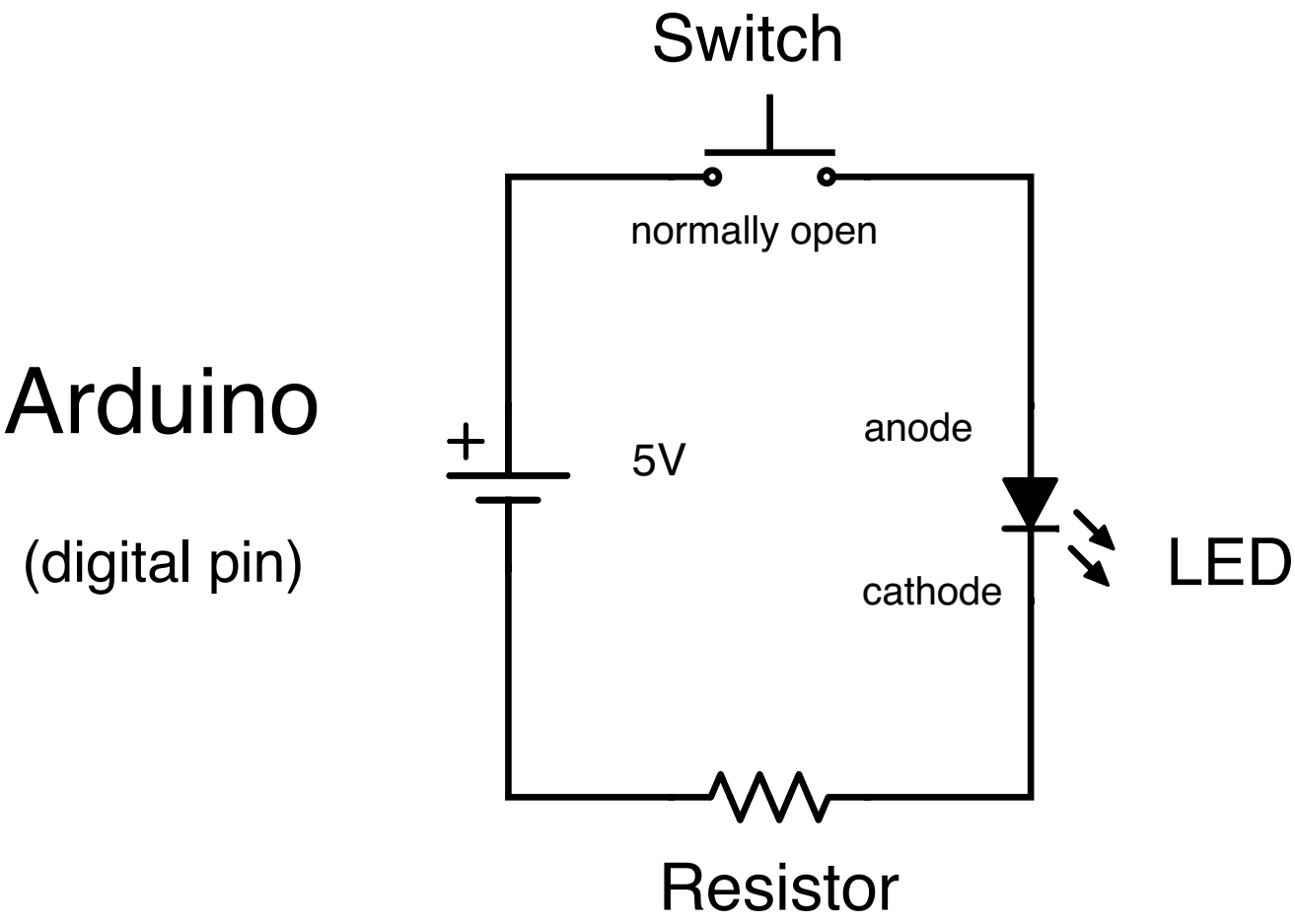
```
// the setup function runs once when you press reset
// or power the board

// declare a new 'byte' type variable and
// set it to the value 8
byte ledPin = 8;

void setup() {
  // initialize digital pin 8 as an output.
  pinMode(ledPin, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(ledPin, HIGH);   // turn the LED on
  delay(1000);                       // wait for a second
  digitalWrite(ledPin, LOW);    // turn the LED off
  delay(1000);                       // wait for a second
}
```

The LED should blink just like it did before! All we've done is create a placeholder called `ledPin` and stored the value 8 to it. Then, whenever we have to refer to the value 8, we can just use the variable `ledPin`. But what is all this `byte` business about? Well that's the variable type. In the programming language that Arduino uses, we have to specify how much memory we want to allocate to memory; in this case, we are allocating one variable with the size of a `byte`. It's like telling the Arduino how much memory we expect to be using; this way the Arduino can check to see if it has enough. Remember the Arduino does not have a lot of memoery.

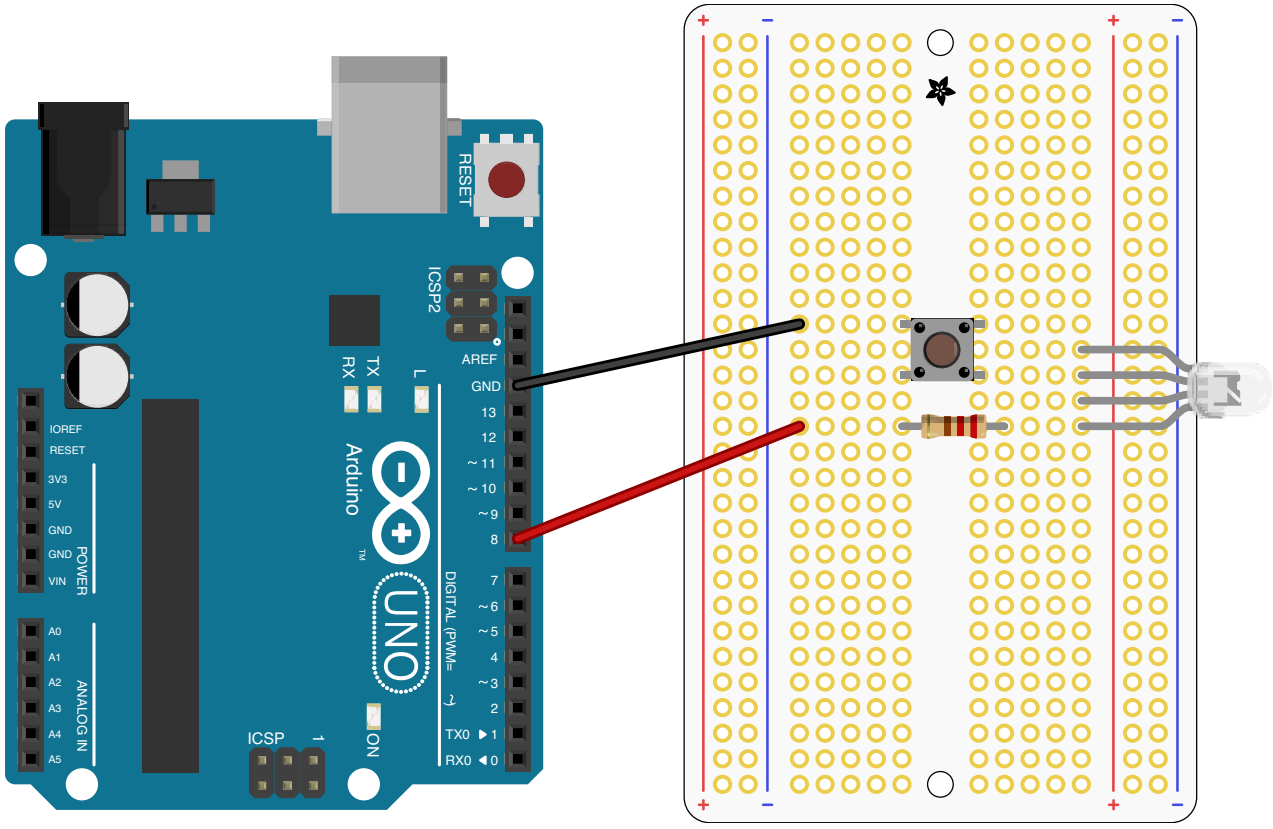
A `byte` stores an 8-bit unsigned number and can range from 0 to 255. Unsigned simply means that it can't be negative; and 8-bit is the same as saying 2 to the power of 8 (which is 255). You might be wondering, what if I want to store the value 438? Well, that wouldn't "fit" into a byte, and you'd have to use an `int` instead. We'll be covering the other variable types in the future, but for now the `byte` should be all that we need.

Part 2 - adding in a switch

Let's make a slight modification to our circuit and add a momentary normally open switch. The way this switch or button works is that unless you press on it, no connection is made accross the terminals (this is known as normally open or NO)



I've updated the layout by adding a switch in the image below and moving the resistor around a bit.



Let's also modify our sketch a little bit so that the LED always stays on; change the `setup()` and `loop()` to look like this:

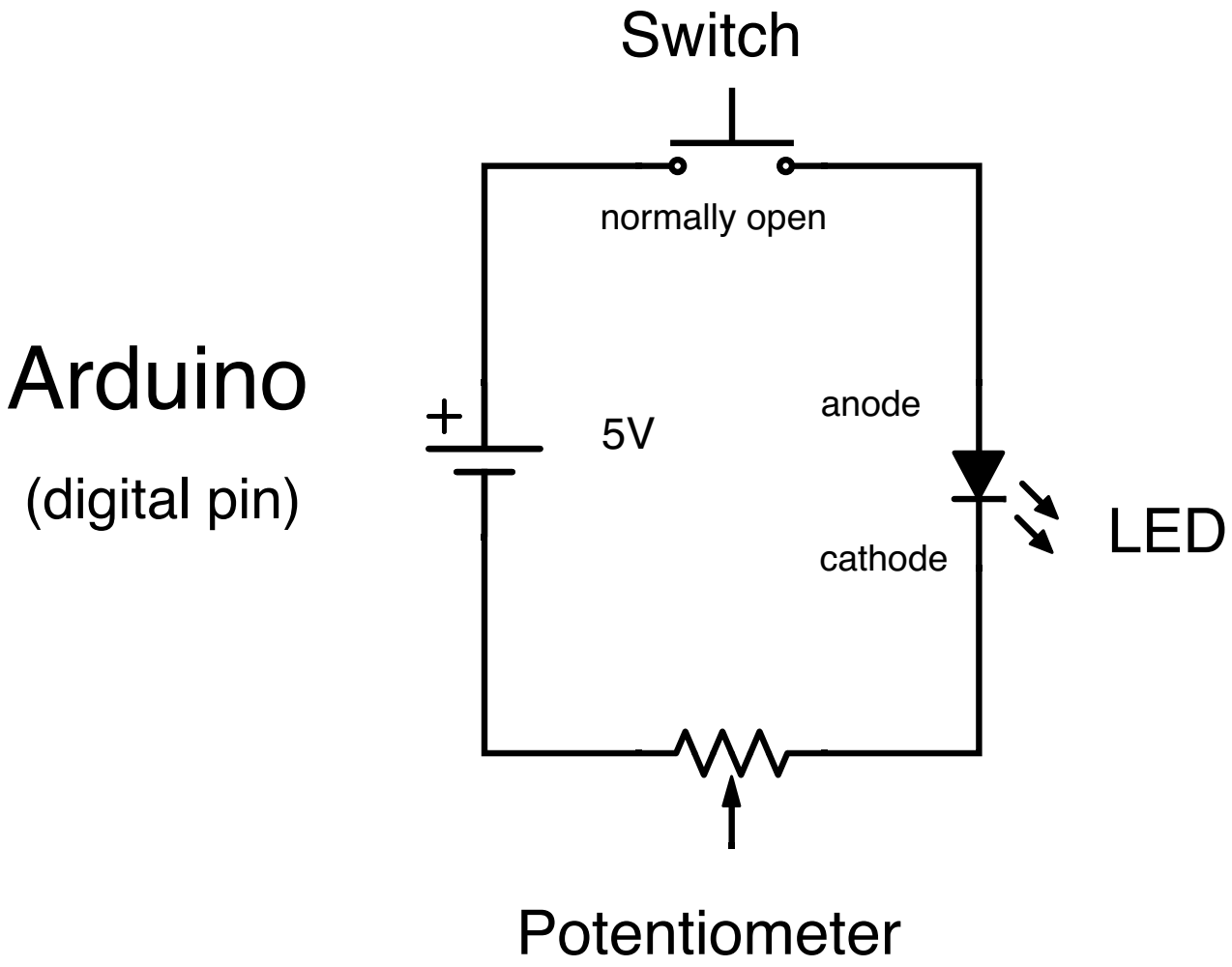
```
// the setup function runs once when you press reset
// or power the board
void setup() {
  // initialize digital pin 8 as an output.
  pinMode(8, OUTPUT);
  digitalWrite(8, HIGH);   // turn the LED on
}

// the loop function runs over and over again forever
void loop() {
}
```

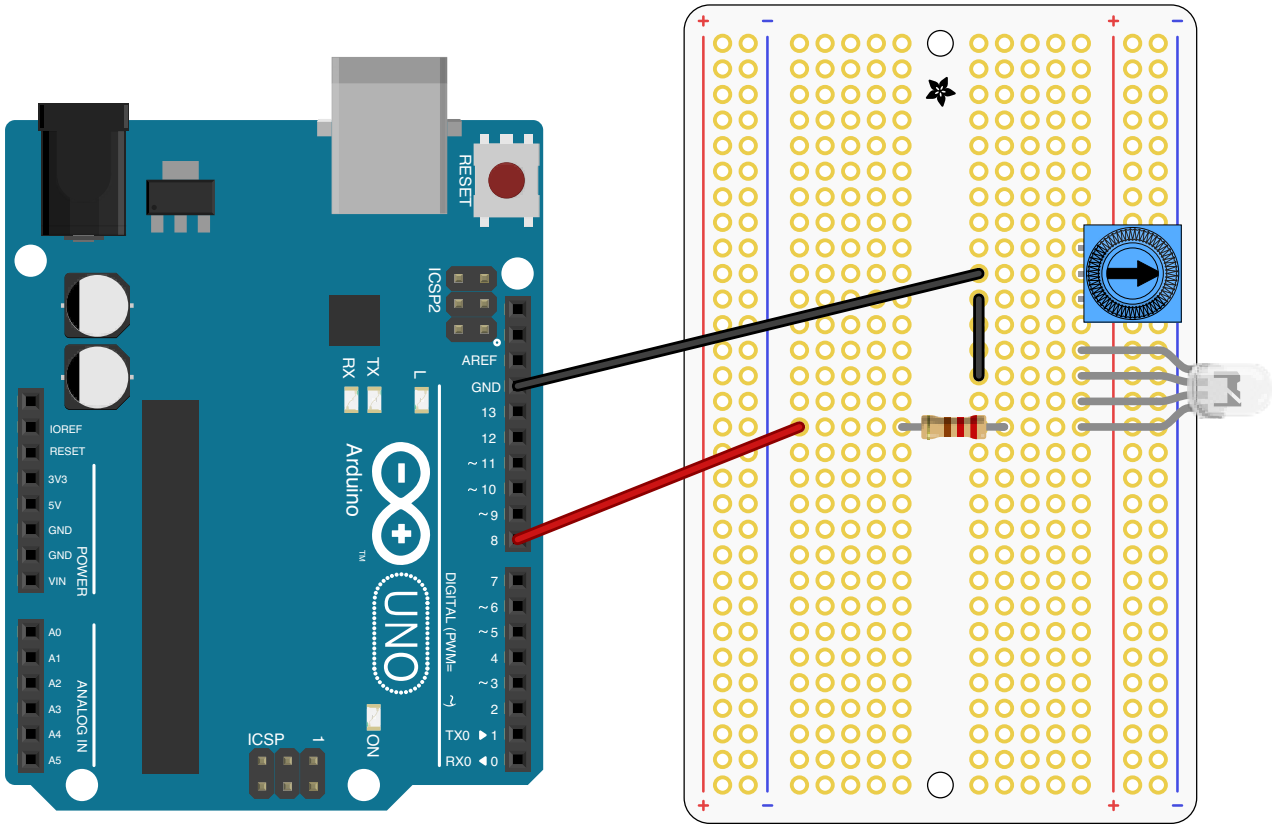
All we've done here is made the `loop()` empty (so that it doesn't do anything) and moved the `digitalWrite()` (which turns the LED on) to the `setup()` function, since we are only interested in turning the LED on once. After uploading the sketch to your Arduino, you should notice that the LED is off unless you press and hold the button!

Part 3 - adding in a potentiometer

Remember in the Introduction where we introduced the concept of a current limiting resistor and how we mentioned that the size of the resistor will impact the brightness of the LED? You might want to re-read that section if you don't remember; in any case, we are going to use a potentiometer (also known as a trimpot) to adjust the brightness of our LED. You can think of a trimpot as a variable resistance resistor; in our case we are using a 10K trimpot which means its resistance varies from 0 to 10,000 ohms.

To use the trimpot, adjust your circuit to match the following:





You can use either sketch that we've talked about (the Blink sketch or the version where the LED always stays on). Update your sketch (if needed), and try twisting the trimpot to adjust the brightness of the LED.

A few important points to remember:

- Adding the trimpot doesn't mean we can take out the other resistor! If we did remove it, and you twisted the trimpot all the way to 0 ohms, you'd blow up your little LED (not literally, but it would break!).
- We only need to use 2 of the 3 pins of the trimpot. The two outside pins will change in resistance, but in opposite directions. So when the left pin is 0 ohms, the right pin will be 10k ohms (and vice a versa).