

<https://resistorcolorcodecalc.com/>

## 1 Introduction

○ Objective:

★ Bonus:

### 1.1 Program

Arduinos are incredibly capable pieces of equipment. The challenge that we often find when programming is that computers do exactly what we ***tell*** them to, not what we ***want*** them to. This can be very frustrating at times. To help avoid confusion, it is beneficial to write down step-by-step what we want the program, or equipment, to do. I'll try to help you by providing this process in the beginning but you will need to do this yourself later. If you run into problems, a great first step for diagnosing issues is to compare what your program/Arduino is actually doing versus what you expected with your written out procedure.

### 1.2 Arduino Requirements

All Arduino programs will have two functions in them. Don't worry about what a function is at the moment, we will discuss them later.

The first function is called **setup**. This runs right at the very beginning when the Arduino first starts up and is used for configuration. The online process we will use to make programs will actually take care of most of this for us. We can typically ignore this for now but it would become very important if we were typing the program instead of using code blocks.

The second function is called **loop**, it runs after **setup** and it runs over and over again in a loop. This is where you will put all of your instructions that you want the program/Arduino to do.



Figure 1: Example of the minimal requirements for an Arduino program.

## 2 LED Basics

An LED, light emitting diode, is an electrical component that produces light. We will be using these to gain a better understanding of circuits and programming.

LEDs have two leads, or wires. The longer lead is the anode and gets connected to the positive side of a power source. The shorter lead is the cathode and is connected to the negative side of a power source. LEDs must always be connected this way. If your LED does not light up as you expect, try reversing it.

LEDs cannot control how much current flows through them so we must always have a resistor in series with the LED. If there is no resistor in the circuit with the LED then something will get wrecked, either the LED or the power source, the Arduino in this case.

### 2.1 Circuit

Try connecting an LED up in a circuit as shown in figure 2.

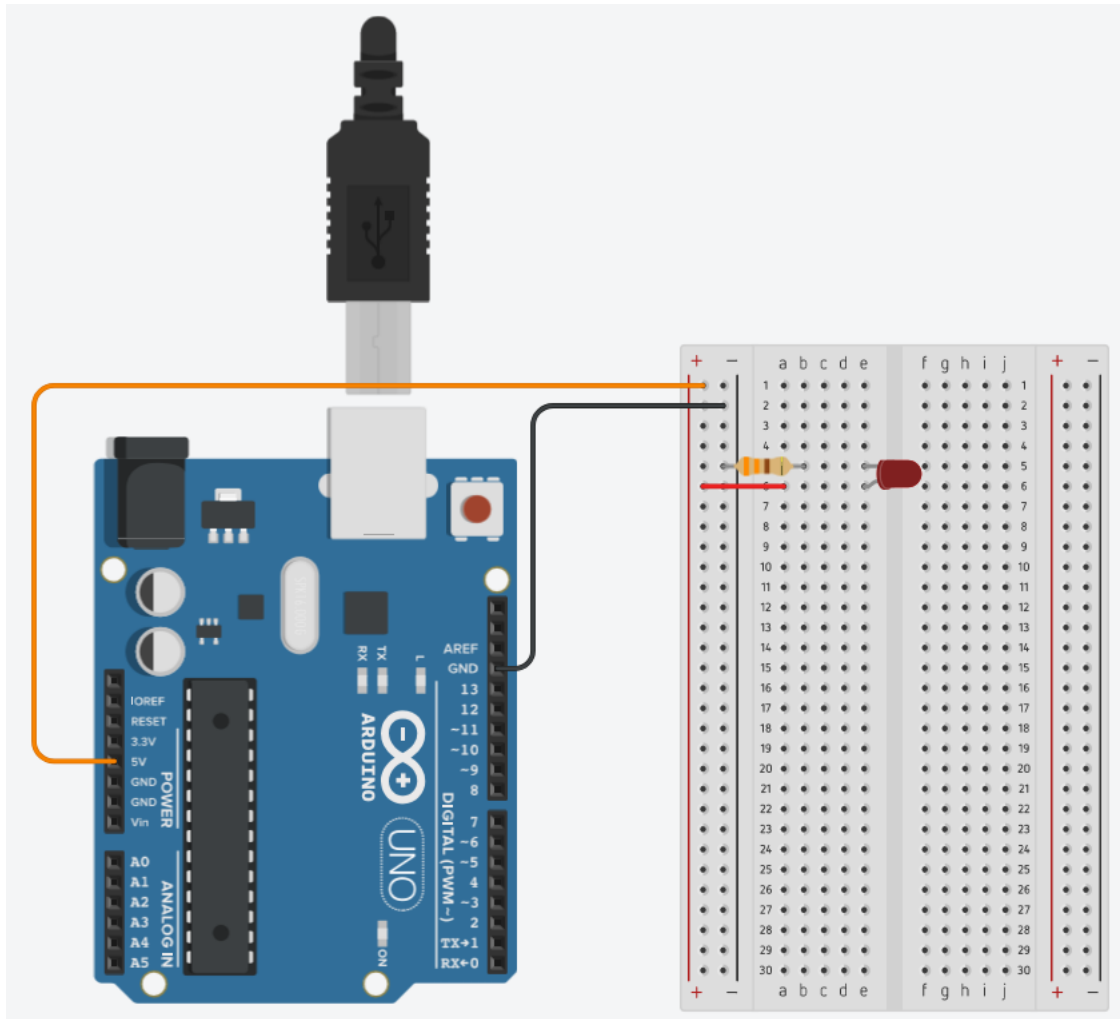


Figure 2: Circuit diagram to directly power the LED.

## 2.2 Program

There is no programming required for this circuit. The Arduino has no control over any of the pins we are using and all it does is provide power.

## 2.3 Objectives

○ Objective: Power an LED.

★ Bonus: What happens when the LED is plugged in backwards?

★ What happens when the order is reversed and the LED is connected to ground and the resistor is connected to power?

## 3 Blink

Now that we have an LED that lights up, lets try changing things up a bit and automate it to blink. Rather than using power from the 5V supply of the Arduino, we will use a digital pin which we can turn on and off.

### 3.1 Circuit

Connect the circuit as shown in figure 3. An important thing to notice as this point is that the LED will now be controlled by pin 11 and not the 5V pin.

Comment on the required resistor value

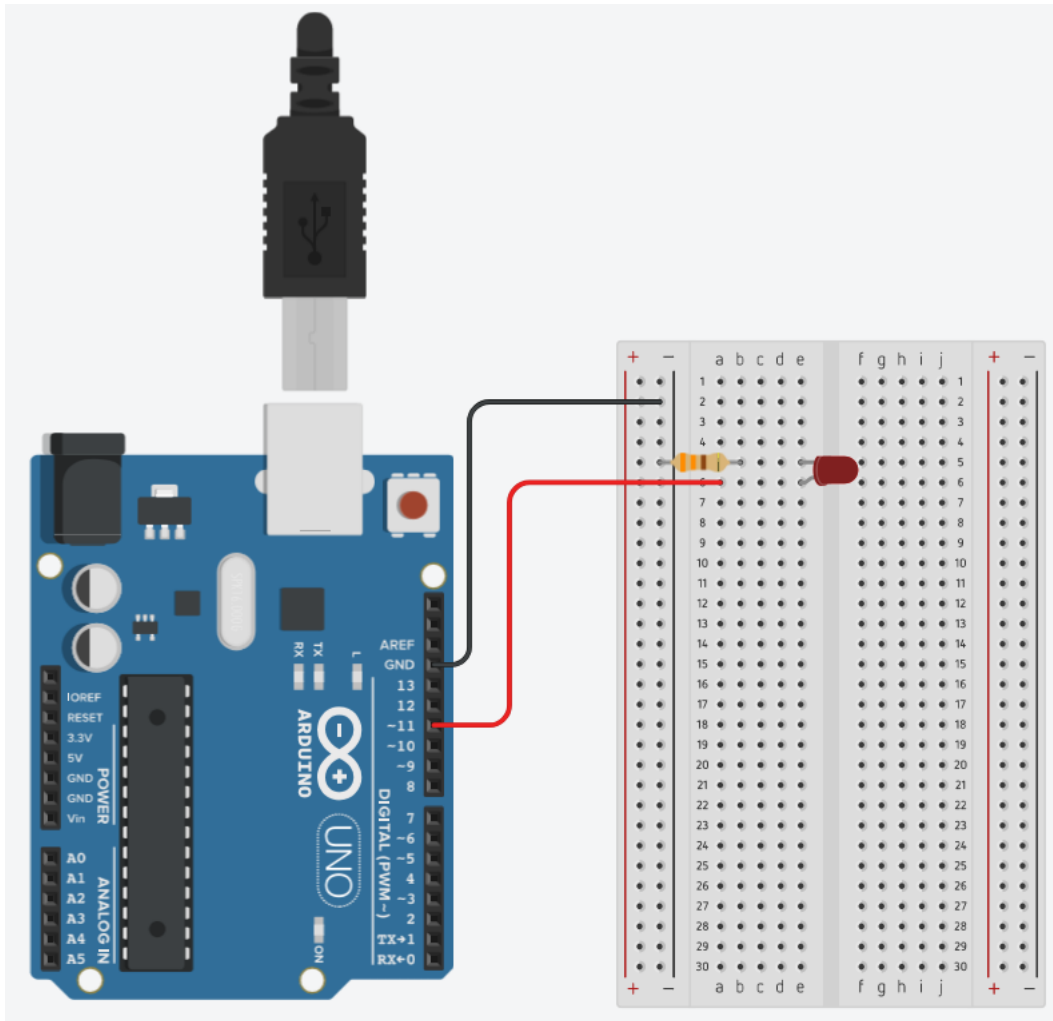


Figure 3: Circuit diagram for the basic LED blink program.

## 3.2 Program

The process for this program is very simple.

1. Turn the LED on by setting pin 11 to ON
2. Delay some amount of time that we want the LED to be on
3. Turn the LED off by setting pin 11 to OFF
4. Delay for the time we want the LED to be off

For this example I will provide you with an example of how to make this process work.

Compare the above process with how the code blocks are setup. In this case, the program is set to wait 500 milliseconds, or half of a second, after the LED is switched on or off.

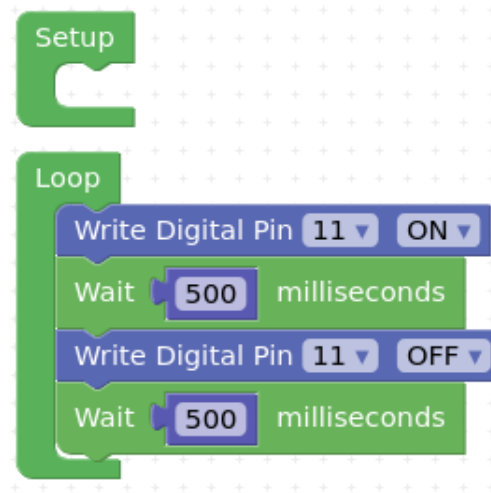


Figure 4: Example of the minimal requirements for an Arduino program.

In the future, I will only provide example code if something significantly new or different is introduced.

### 3.3 Functions

Functions are a simple way to take a specific process, like the previous steps to turn an LED on and off, and make it easy to use. Functions also help make code easier to read, build, maintain and fix.

Look at this example. It has the exact same process and outcome as our blink program except the steps to blink an LED are in a function and we refer to that function when we want the blink process to run.

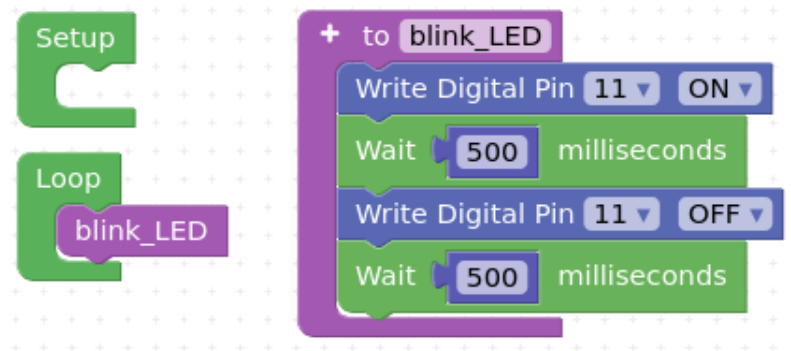


Figure 5: Example of the minimal requirements for an Arduino program.

Do you see how this is easier to read? When the Arduino is running it will **blink the LED**. If we want to see the exact process of how Arduino controls the LED and causes it to blink, we can look at the function **blink\_LED**.

Imagine if we had 3, 10, or even a 100 LEDs to control. The **loop** function would get very confusing very quickly!

It should be noted that you have been using functions already, you just didn't realize it. Both **Write Digital Pin** and **Wait** are functions that are built-in to the Arduino system that you always have access to. These functions also have required input values. For **Write Digital Pin** you must provide the pin number that you want to make a change to and the new value that you want it to have. These functions help to make your life much easier as you do not need to understand what they are doing on a step-by-step level, you only need to know what they accomplish in the end.

### 3.4 Objectives

○ Objective: Program the Arduino to switch the LED on and off.

★ Bonus: What happens when you vary the amount of time in each of the delay blocks?

★ Reduce the time in each delay block to be less than 15 microseconds each. Can you still see the LED flash?

★ Keep the total sum of time in each delay block to be 15 microseconds while changing the time in each delay block. For example, 3 and 12 microseconds? 5 and 10? 7 and 8?

12 and 3?

## 4 Traffic Lights

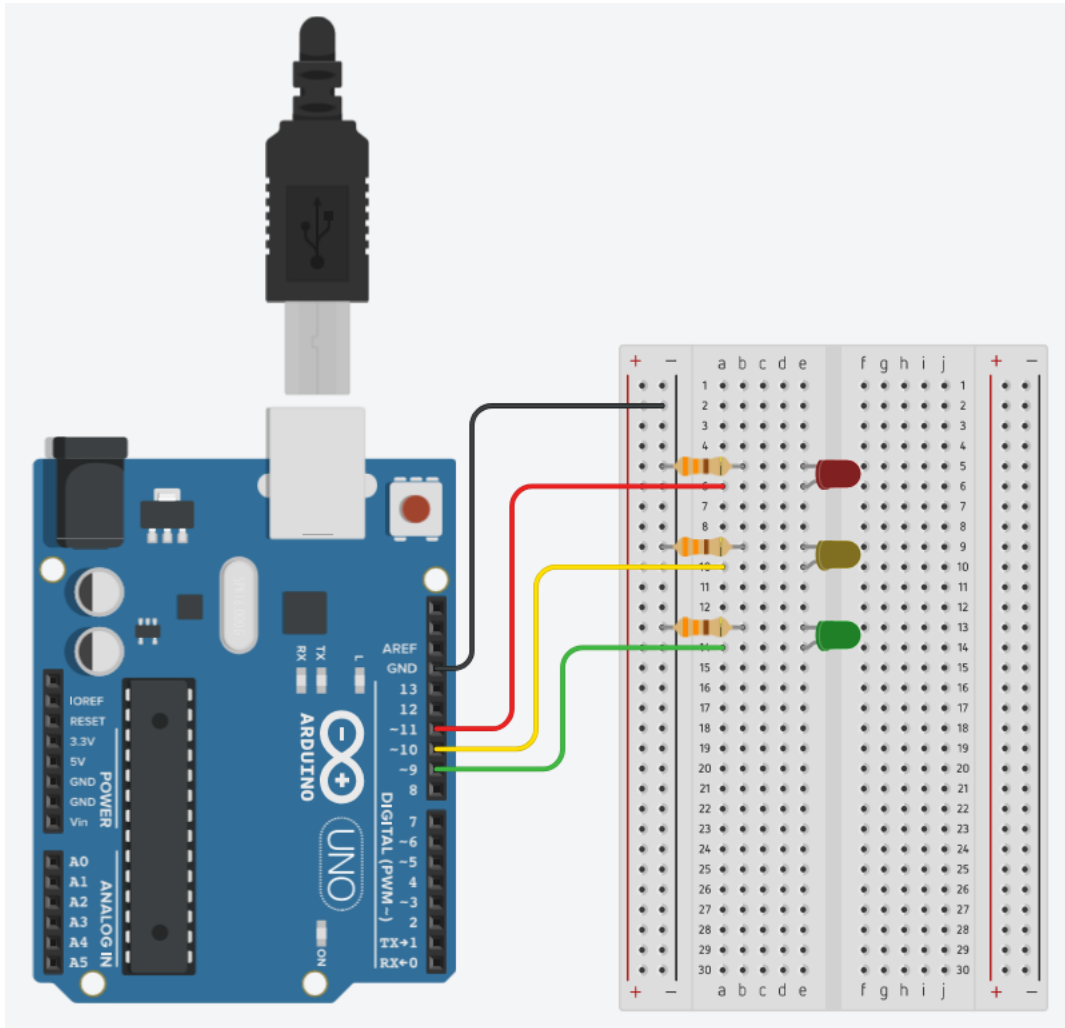


Figure 6: Circuit diagram for the traffic lights program.

### 4.1 Process

Traffic lights have a very simple process that they run through.

1. Go - red off & green on



2. Caution - green off & yellow on
3. Stop - yellow off & red on

## 4.2 Programming

Using the program you made to blink an LED (use the version which has the `blink_LED` function) as an example, try to make a new program for the traffic lights. Make a function for the red LED, another for the green and a third for the yellow LED.

Do you get the result that you are expecting? Remember, real traffic lights never have all 3 lights off at the same time. This would cause confusion and people may not know what they are supposed to be doing if no light is on. Do you always have one LED being lit up?

Is this a good time to introduce buttons?

## 5 Controlling LED Brightness

Controlling the brightness of an LED can be done in two different ways. The first way to change the amount of current flowing through it using a resistor. The second method is actually a trick played on your eye, but we'll discuss this more in a bit.

### 5.1 Resistors: Reducing current flow

As stated at the beginning, LEDs cannot control the amount of current flowing through them. To ensure that they don't get damaged by too much current flowing through them we use resistors to limit the current flow. We can use this to our advantage to change how bright an LED is since the more current that flows through an LED, the brighter it is.

The following circuit uses a potentiometer, or pot, which is simply a resistor that we can change the resistance of. We will leave the resistor in the circuit that we were using before in place and add the potentiometer.

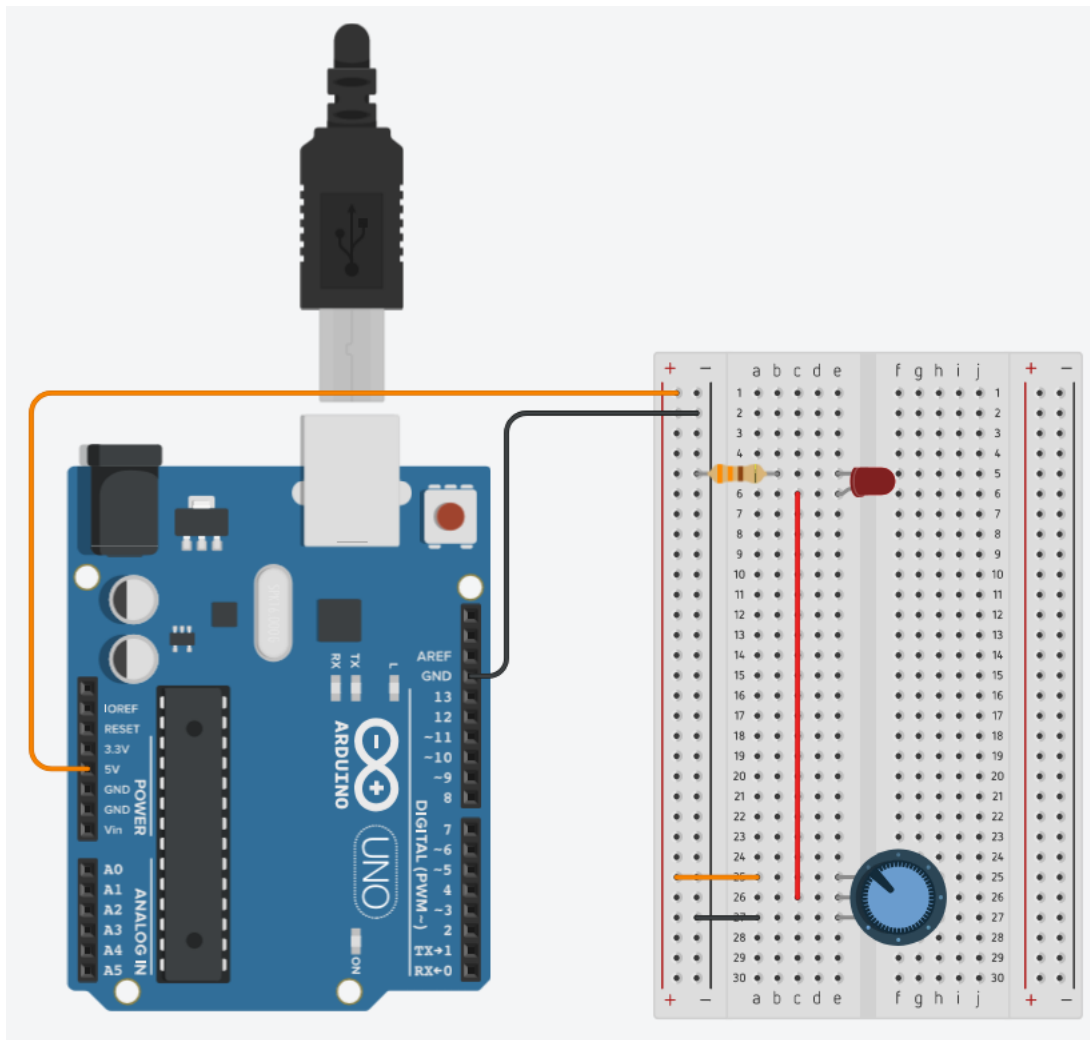


Figure 7: Circuit diagram for the traffic lights program.

The circuit uses the Arduino to power the LED only and no programming is required.

When you turn the dial on the potentiometer, the LED will get brighter as the resistance of the potentiometer decreases and more current flows. If you turn the dial the other way, the resistance of the potentiometer will increase, reducing current flow and the LED will get dimmer.

Is this a good time to do the Analog read process to show how LEDs turn on at specific voltages? Serial communication introduction?

## 5.2 PWM: On/Off really fast!

The previous method used hardware and actually caused the LED to get brighter and dimmer. This second method will use software to achieve the same goal but this time it will only make the LED appear brighter or dimmer using an optical trick.

### 5.2.1 Circuit

First lets set up the circuit. You'll notice that we are still using the potentiometer. However, rather than using the potentiometer to directly control the LED, we will use the Arduino to make a measurement of the potentiometer's value.

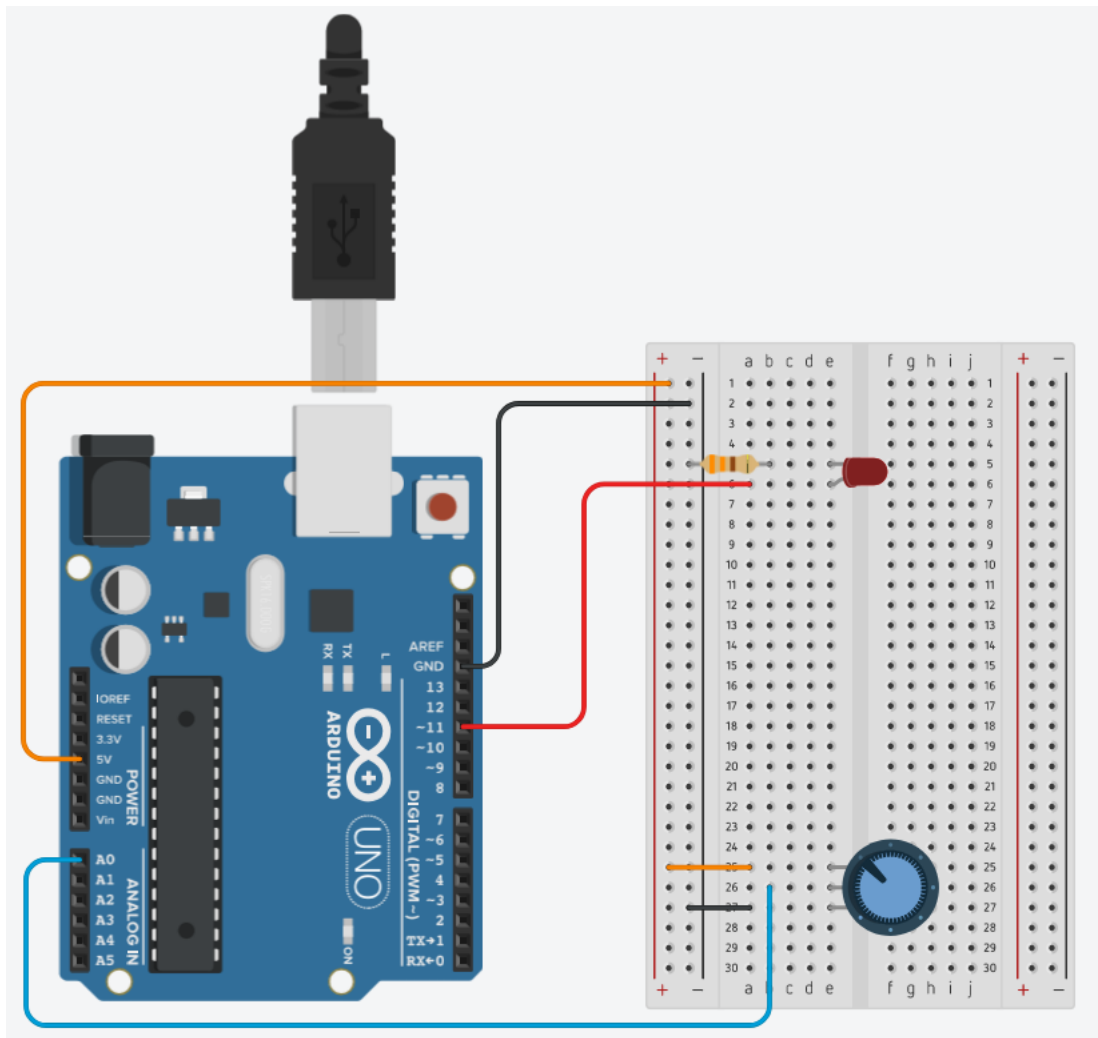



Figure 8: Circuit diagram for the traffic lights program.

### 5.2.2 PWM Signals

On the Arduino board you will see a little  mark next to some of the pin numbers. This indicates that the pin can be used to output a Pulse Width Modulated signal, or PWM signal. It sounds more complicated than it is.

Remember at the beginning we made a program to blink the LED on and off repeatedly? That basically all a PWM signal is, turning a signal on and off. The difference is that we are now going to turn the light on and off really, really fast. Like, 480 times every second

fast. The only control that we have over the PWM signal is the percent of time that the signal is on vs off.

### 5.2.3 Program

The process that we need for this program is:

1. Read the value of the potentiometer using **Read Analog Pin**
2. Optional: Send the potentiometer value back to the computer using serial so we can see what the Arduino knows
3. Use the **Map** function to get an acceptable PWM value
4. Update the PWM signal that is output using **Write Analog (PWM)**

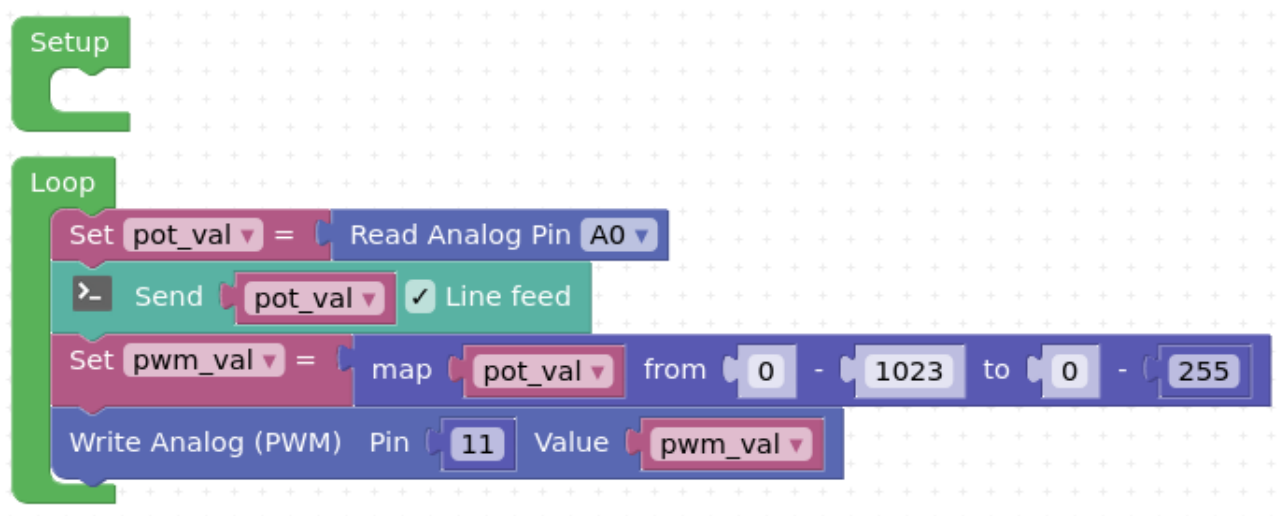


Figure 9: PWM LED brightness control program.

### 5.2.4 Map - Built in Function

Note: How the **Map** function works is not important at this time. You are free to skip this section and simply use the values that are provided in the program we make for this section.

**Map** is a built in function that Arduino has by default and it requires some input values and it returns a result. We don't need to know how it works specifically, we just to know what it does over all. The inputs of **Map**, in order, are:

- value - A number that we to convert from one range A to range B
- fromLow - The lower end of range A
- fromHigh - The upper end of range A
- toLow - The lower end of range B
- toHigh - The upper end of range B



Figure 10: PWM LED brightness control program.

**Map** is a function that returns a value. In the example above, we are going to store the result of **map** in a variable called **new\_value**.

For example, lets say you take a test and get a score, value, of 10. The minimum score, fromLow, that you can get is 0 and the maximum score, from High, is 20. We can **Map** your score to a percent with a minimum percent, toLow, of 0% and a maximum percent, toHigh, of 100%. **Map** will return a value of 50.

## 6 Serial Communication

A helpful diagnostic method.