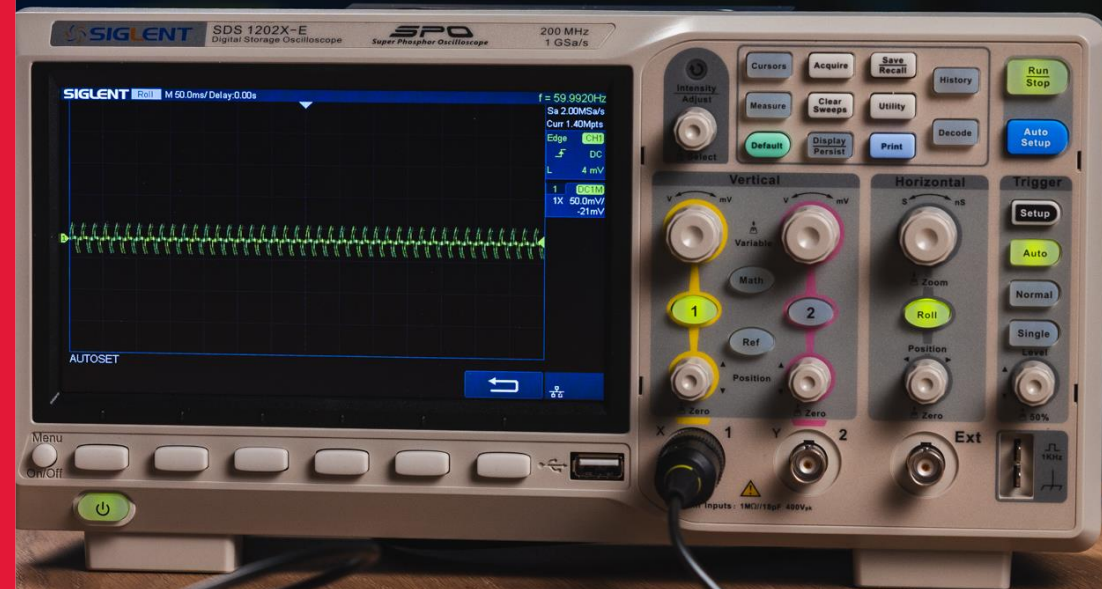


Makerspace Orientations: Basic Electronics and Circuitry

Wiring, Circuits, Arduino and more

JACOB TUROLA, LANA YUAN

libraries | **YORK** 





Expectations for

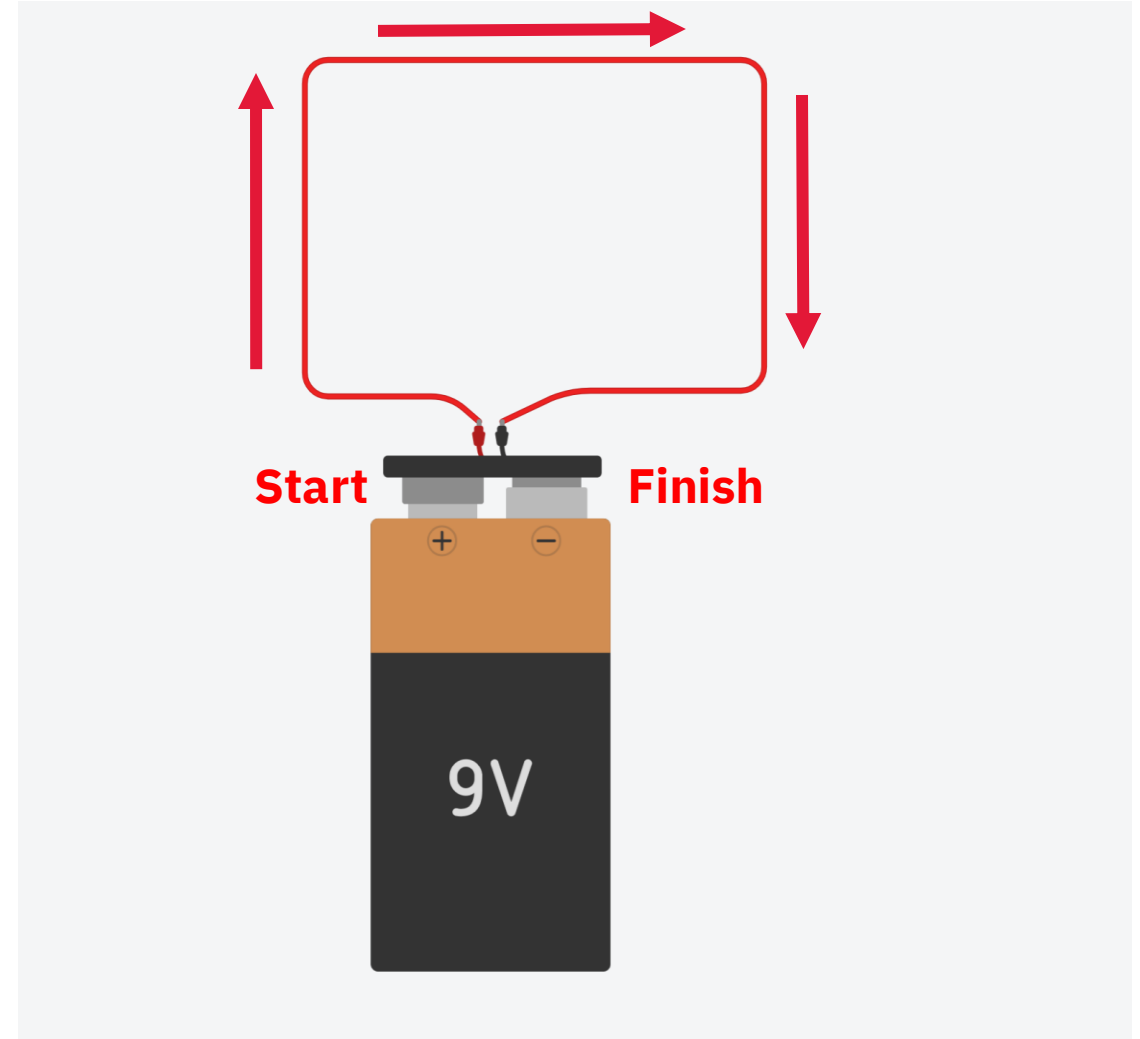
- › Learn how a circuit works
- › Basic circuit drawings
- › What is arduino
- › Basic electronic components
- › Basic coding
- › Serial monitors
- › Breadboard prototyping

Follow Along!


- You'll probably notice we left you some equipment at your table. Feel free to follow along and experiment with the gear provided!
- If anything breaks or doesn't work, let us know

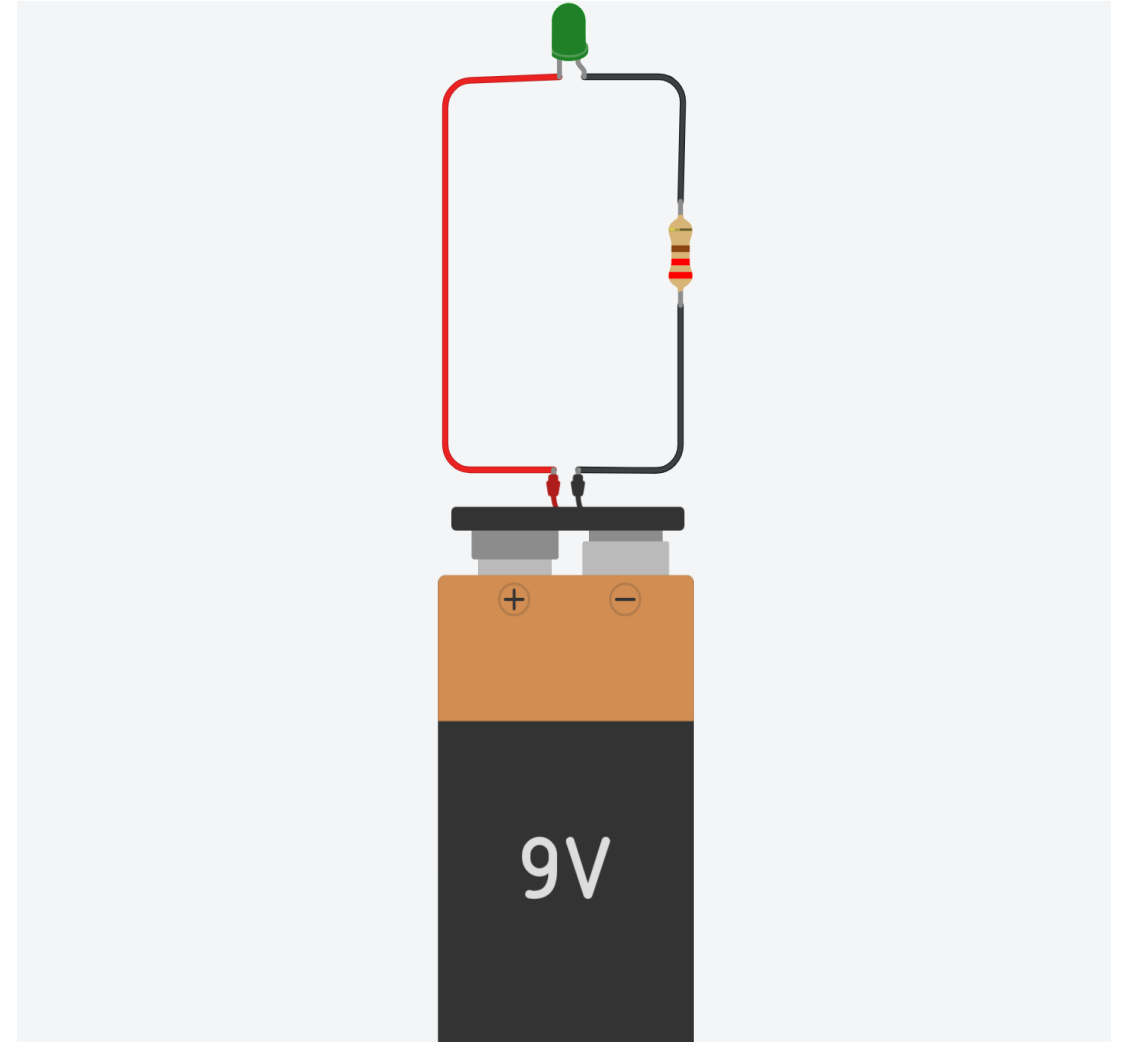
What is a Circuit

- In the simplest terms, a circuit is a continuous route for electricity to travel through a system
- A simple analogy is that electricity can be seen as cars on the road whereas the wires or other conductive material is the road itself
- Wires define the path that the road takes
- Every circuit needs a start (+) and End (-)
- Electricity will flow from one end of the circuit to the other



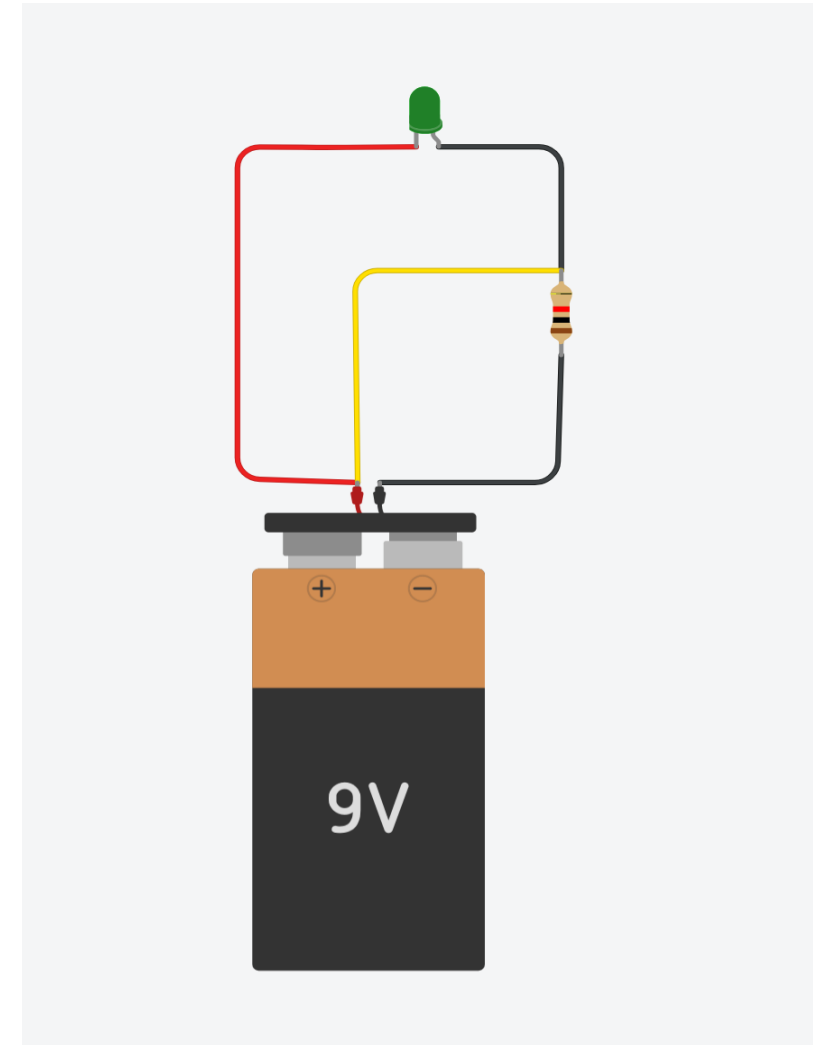
Circuits Continued

- In a circuit, anything that uses electricity for a function is considered a "load"
 - Ex: Lights, motors, sensors, etc
- In the circuit, electricity passes through them and continues down the path
- Other parts resist the flow of electricity and thereby reduce the power output
 - Called resistors 
- In the car analogy, loads are drive throughs and resistors are speed limit signs



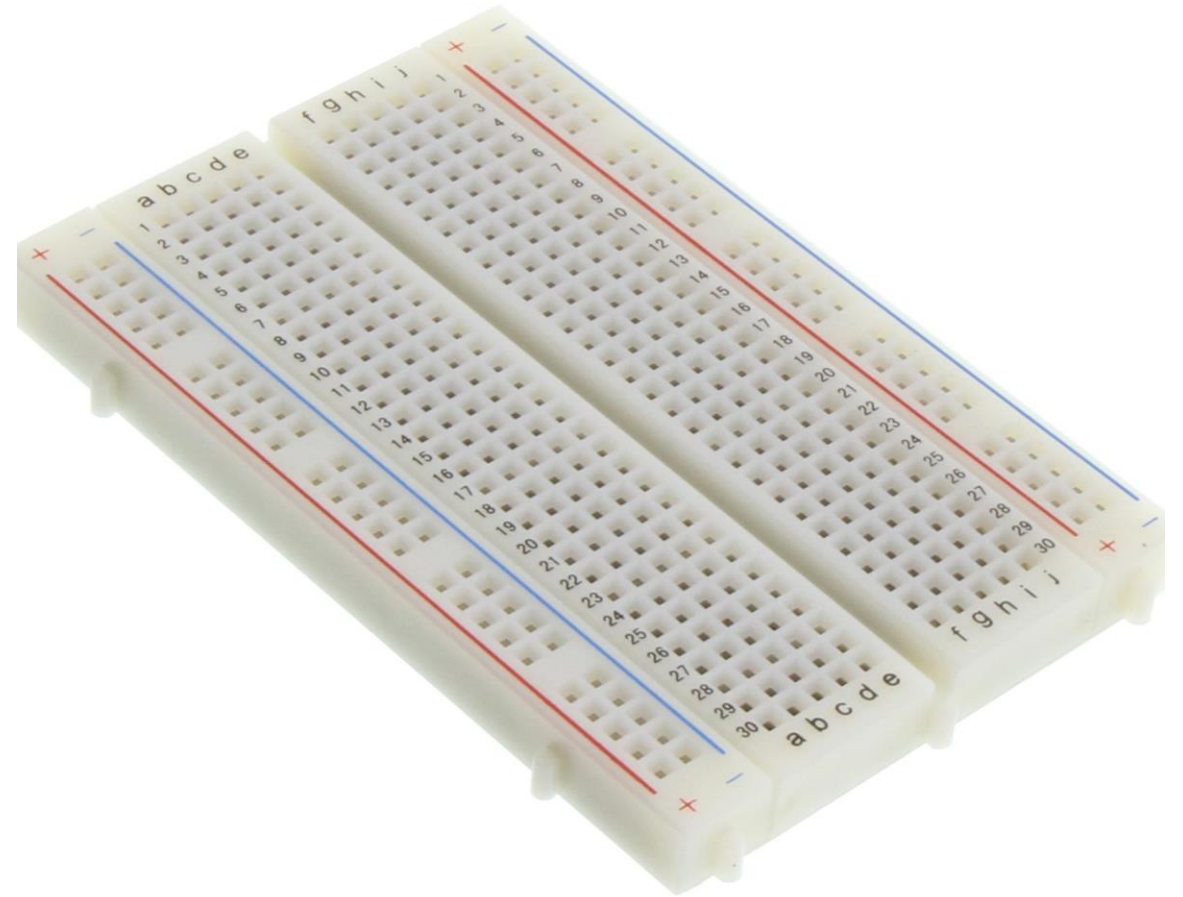
Short Circuits

- A short circuit is when there is a shortcut made between + and – ends of a circuit that bypasses the load
- If this happens electricity will follow the path of least resistance and surge through the shortcut
- This can break components, cause fires or overload sensors.
- Avoid them as best as you can. Thankfully, the arduino has protection against some short circuits and the USB port plugged in also has surge protection



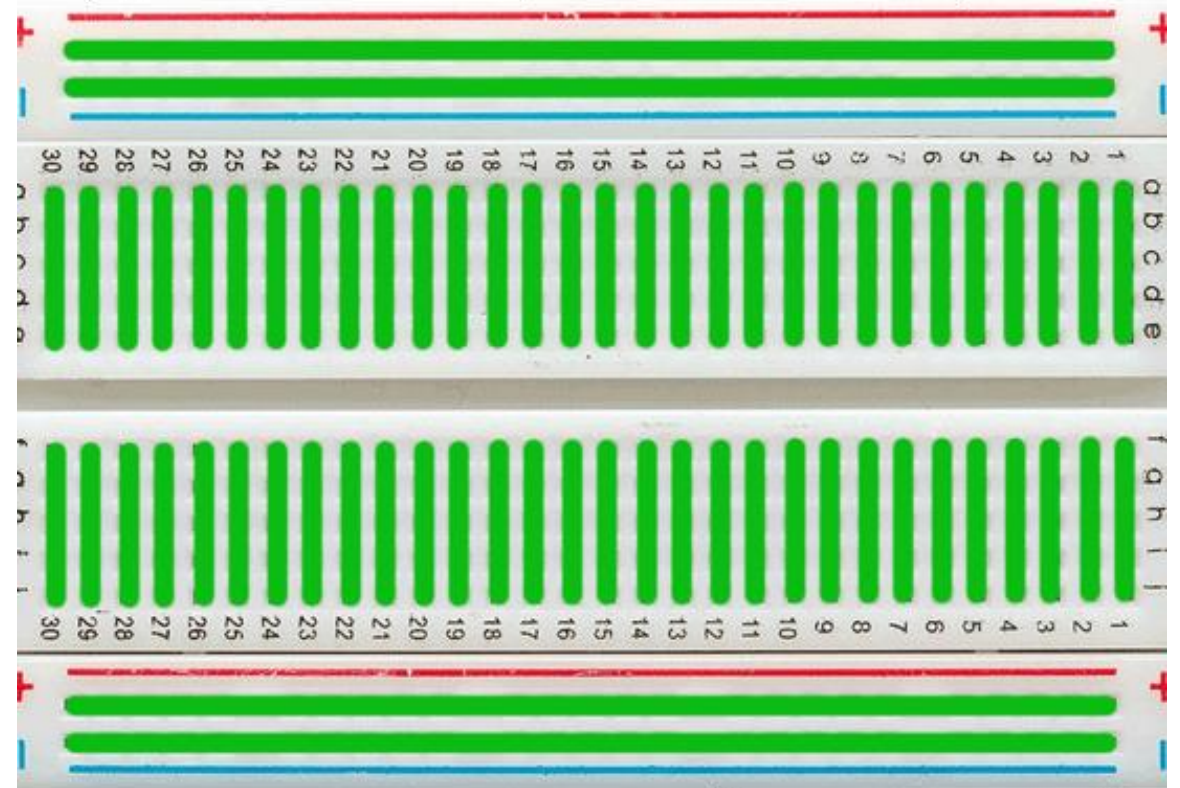
Breadboards

- A breadboard is a platform to prototype a circuit design on
- It has many holes to insert wired components into and has metal underneath each hole that connects each row
- Every row is considered a separate "wire"
- One of the most widely used prototyping methods for electronics
- You use it by "Plugging" wires or metal ends into the holes



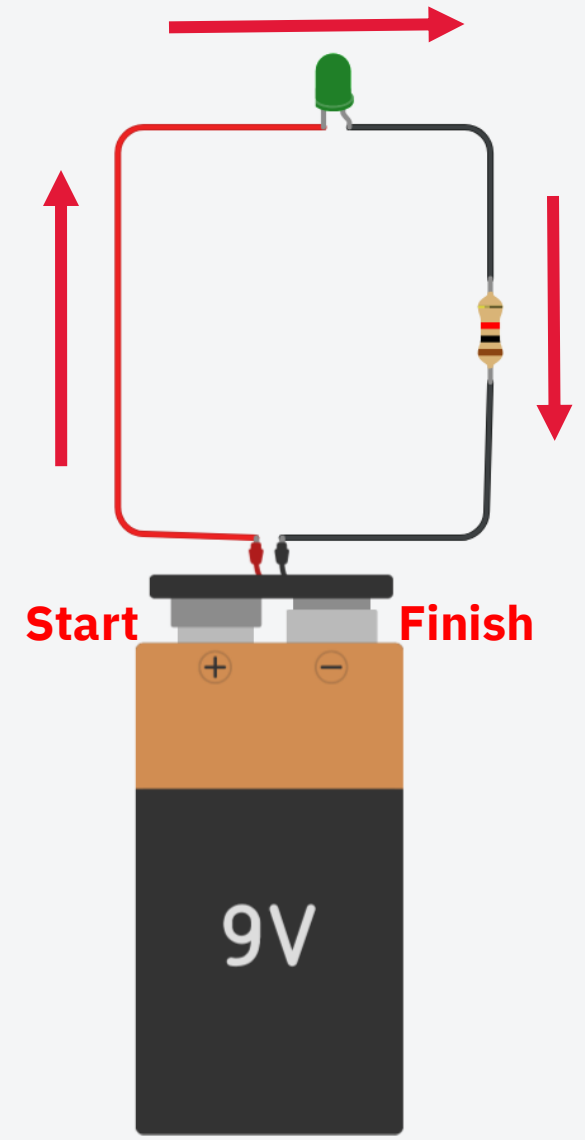
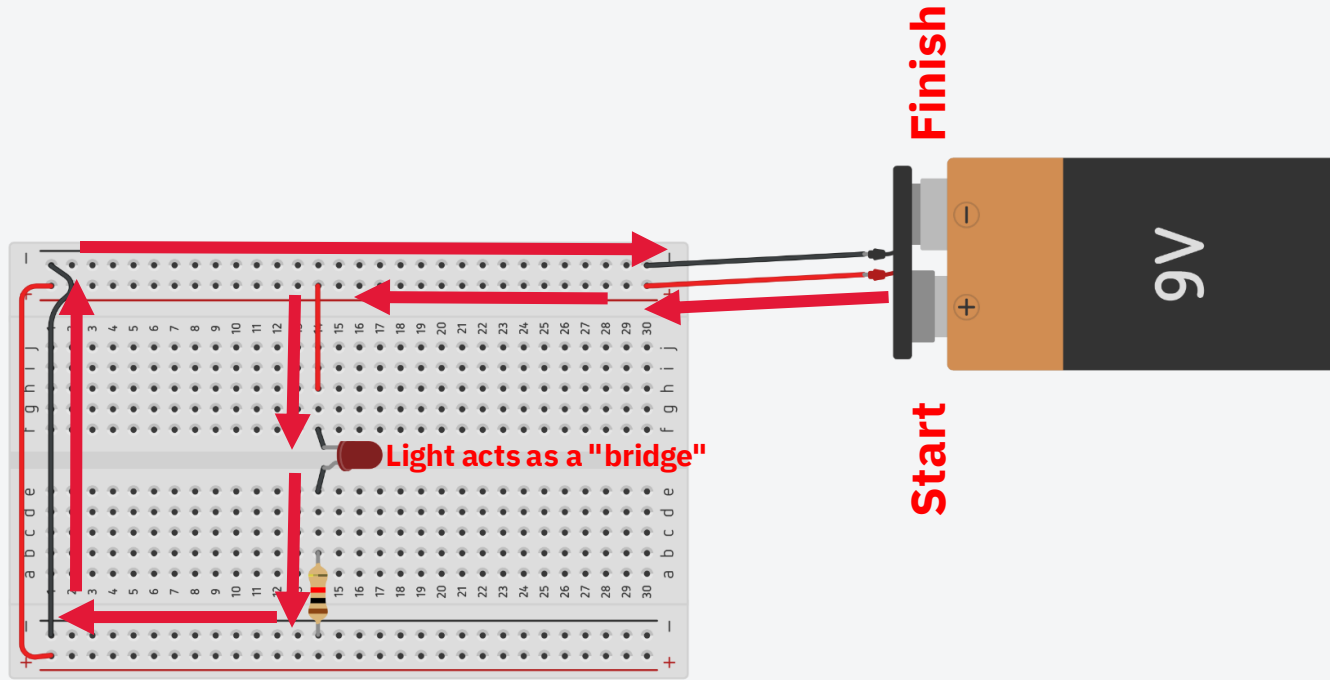
Breadboards Continued

- Each row is separated and is a separate "wire"
- Highlighted here are all of the distinct areas on the breadboard
- It is important to remember that electricity will only flow within each separate section if power is brought to it
- You can connect each section via wires or other components like LEDs or resistors

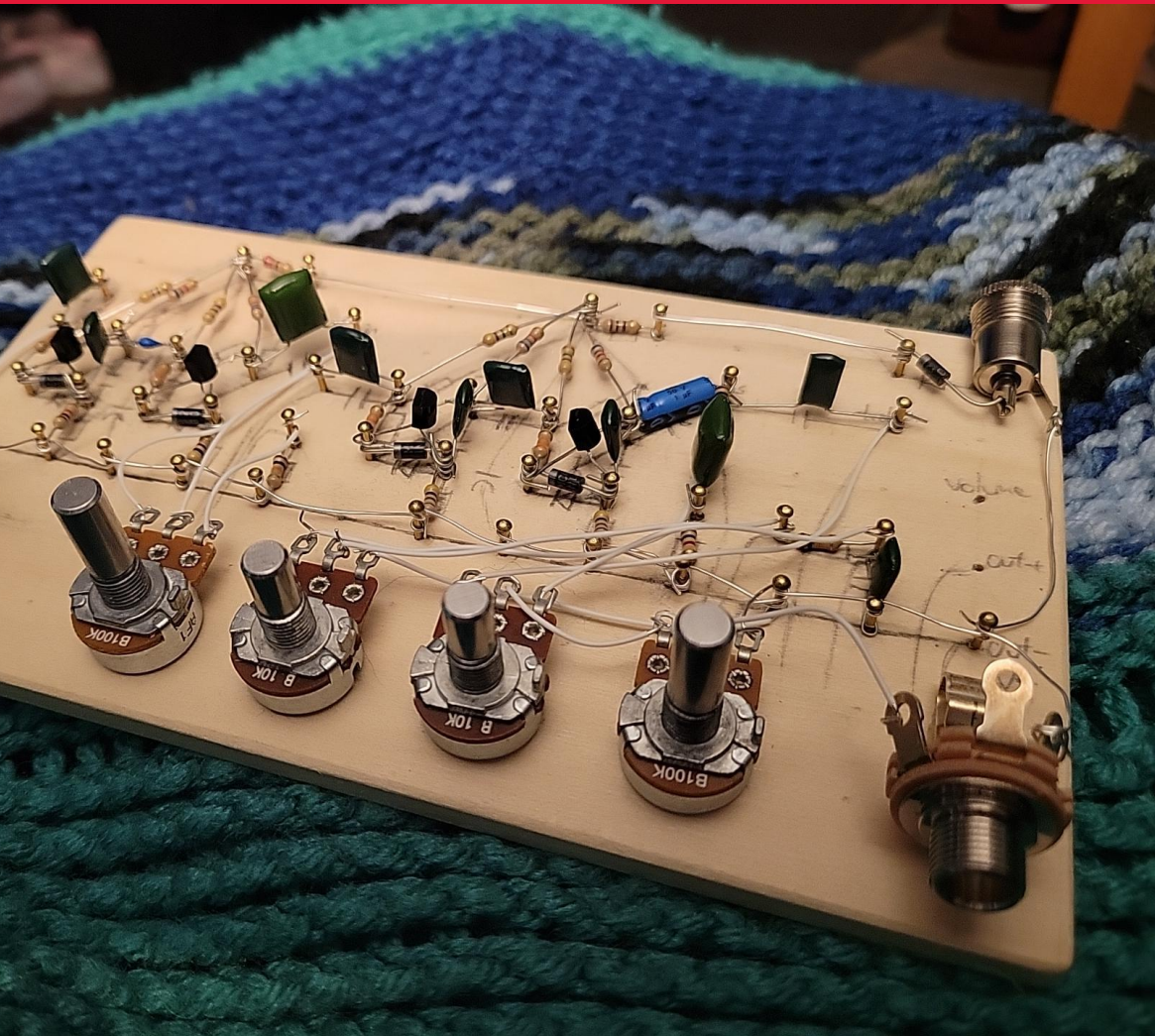


Circuit View on a Breadboard

These two circuits are functionally the same

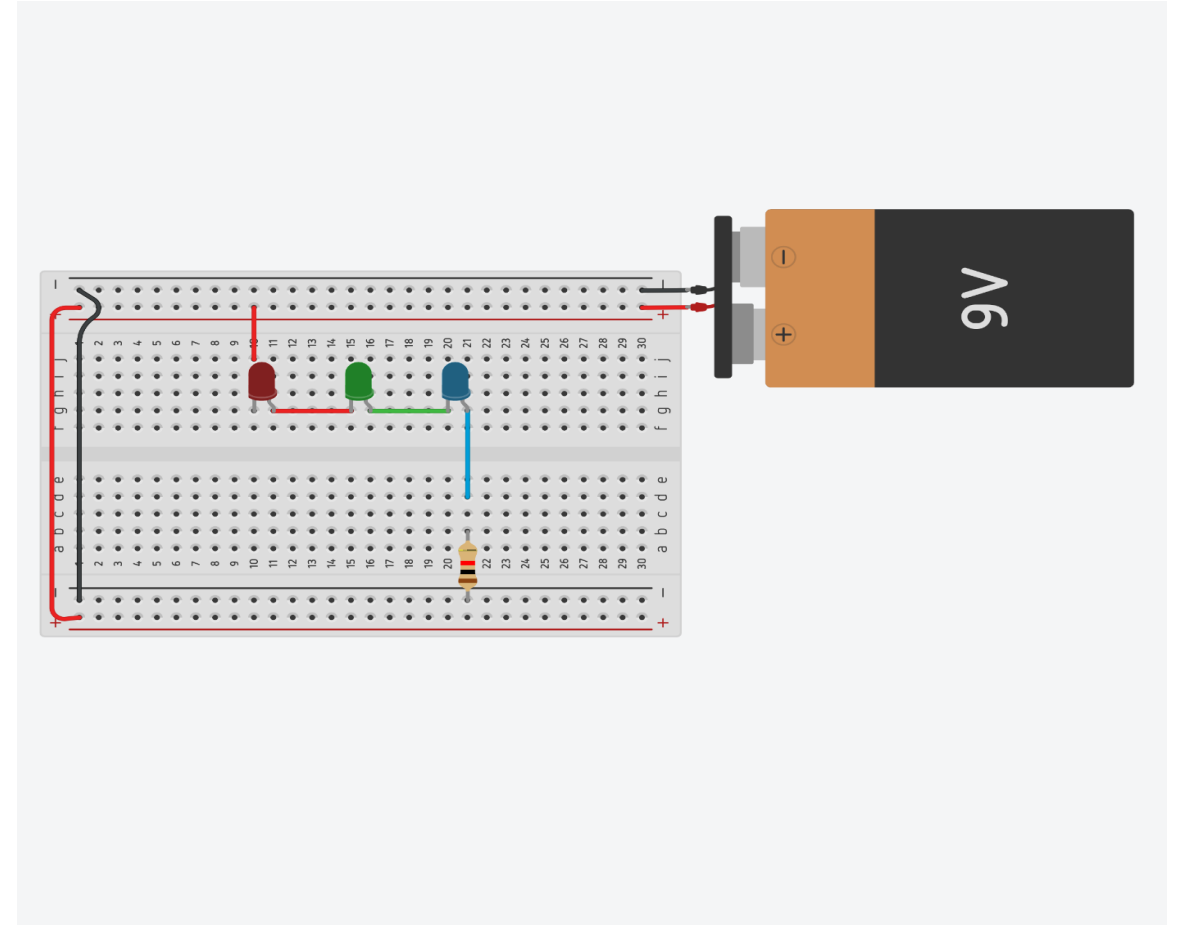


Old School Breadboards



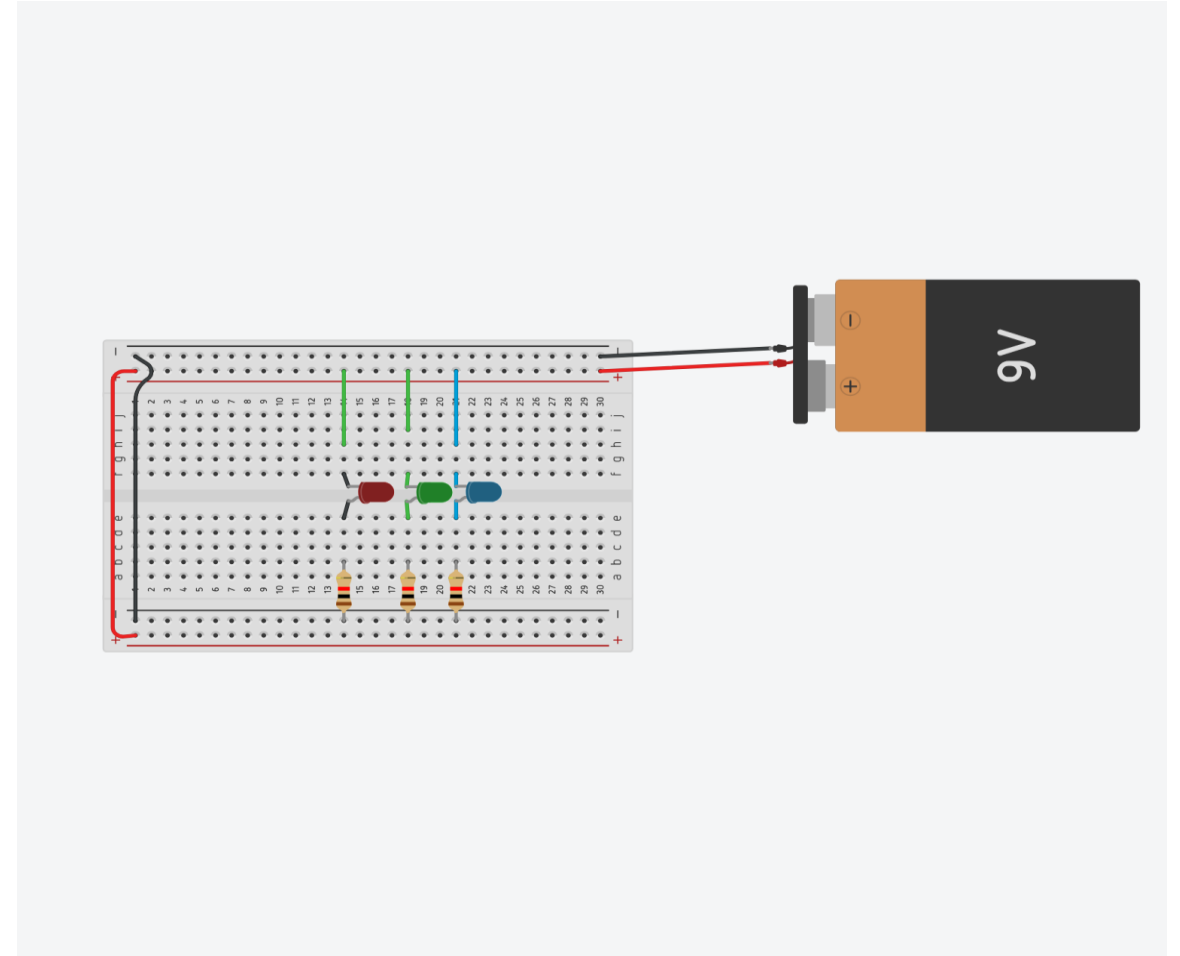
Series Circuits

- Series circuits are a way of assembling that place all loads in a line
- This is an easier way to assemble things
- A good advantage of this method is its simplicity
- However, if one breaks: All components will not function as the continuous line from + to – has been cut off



Parallel Circuits

- Parallel circuits are a way of assembling that place all loads in separate lines that all connect to the same endpoint
- This is an efficient way to organize a circuit
- A good advantage of this method is that each load is independent
 - If one breaks, the rest continue fine
- However, it means that the energy source is split across the circuit

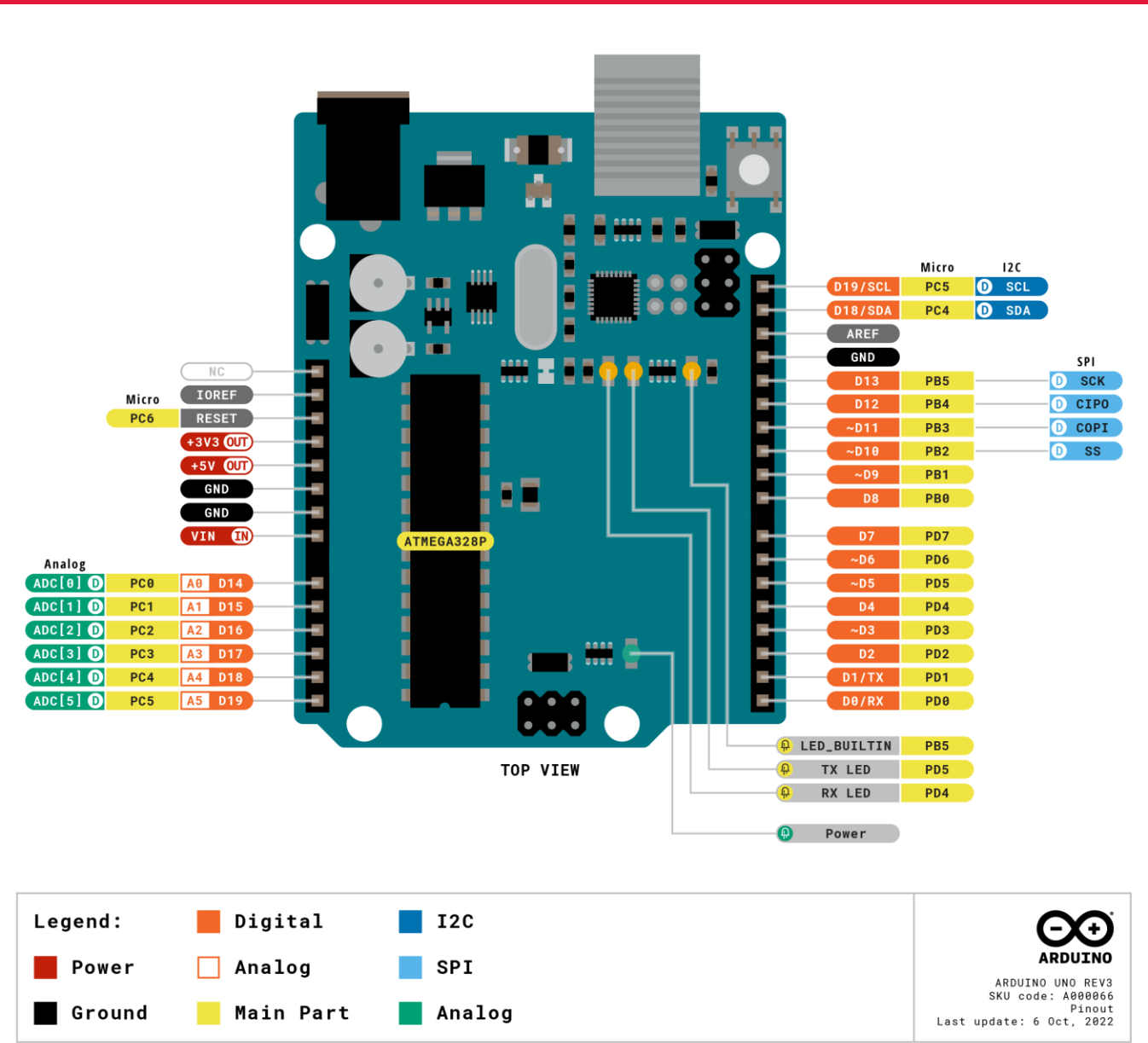


Arduino

- This is a circuit board designed for prototyping
- Designed to make programming and custom circuitry easier
- Looks complicated but is a rather simple device
- It takes simple inputs and outputs as you deem it fit
- We'll be going over how to code it later
- What's important is that it has **digital pins, power output pins (+) and ground pins (-)**

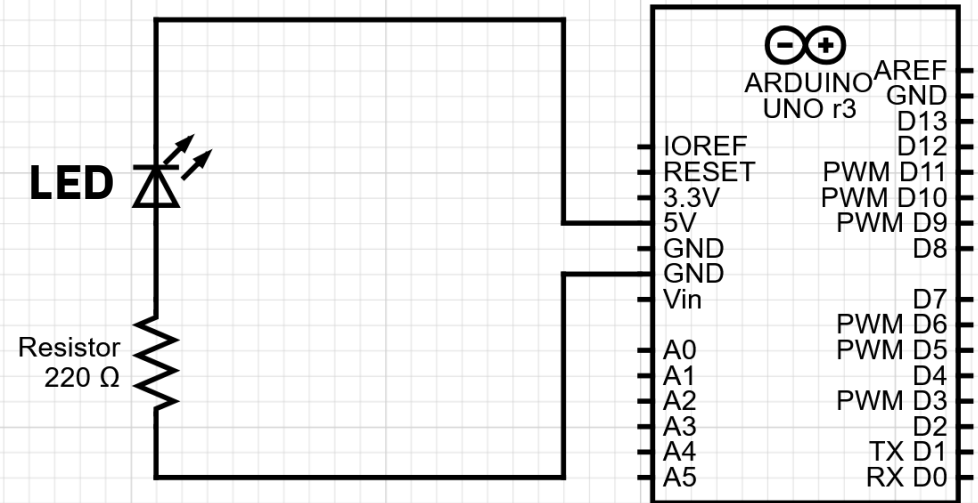


Arduino Board Diagram

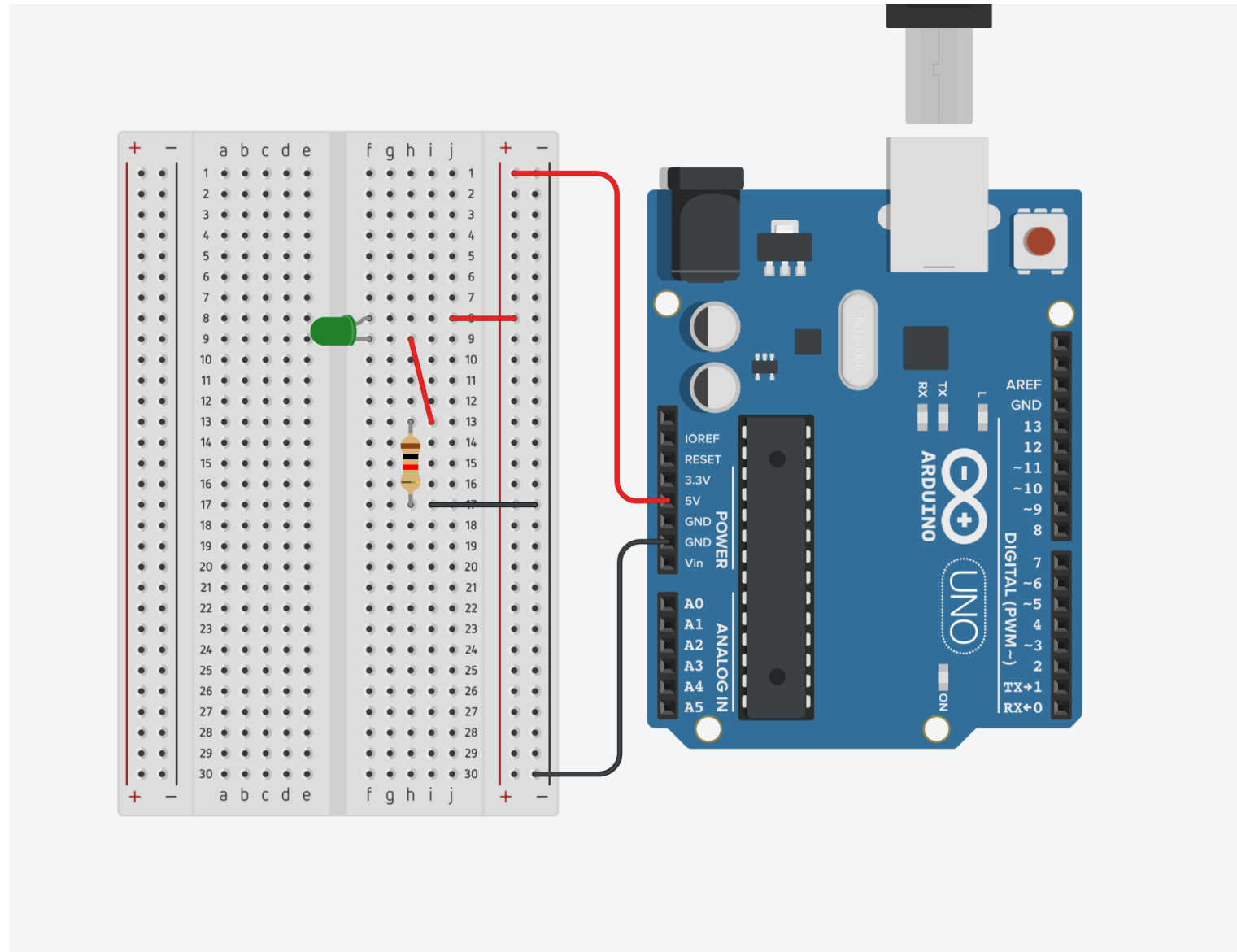


First Circuit Puzzle: LED

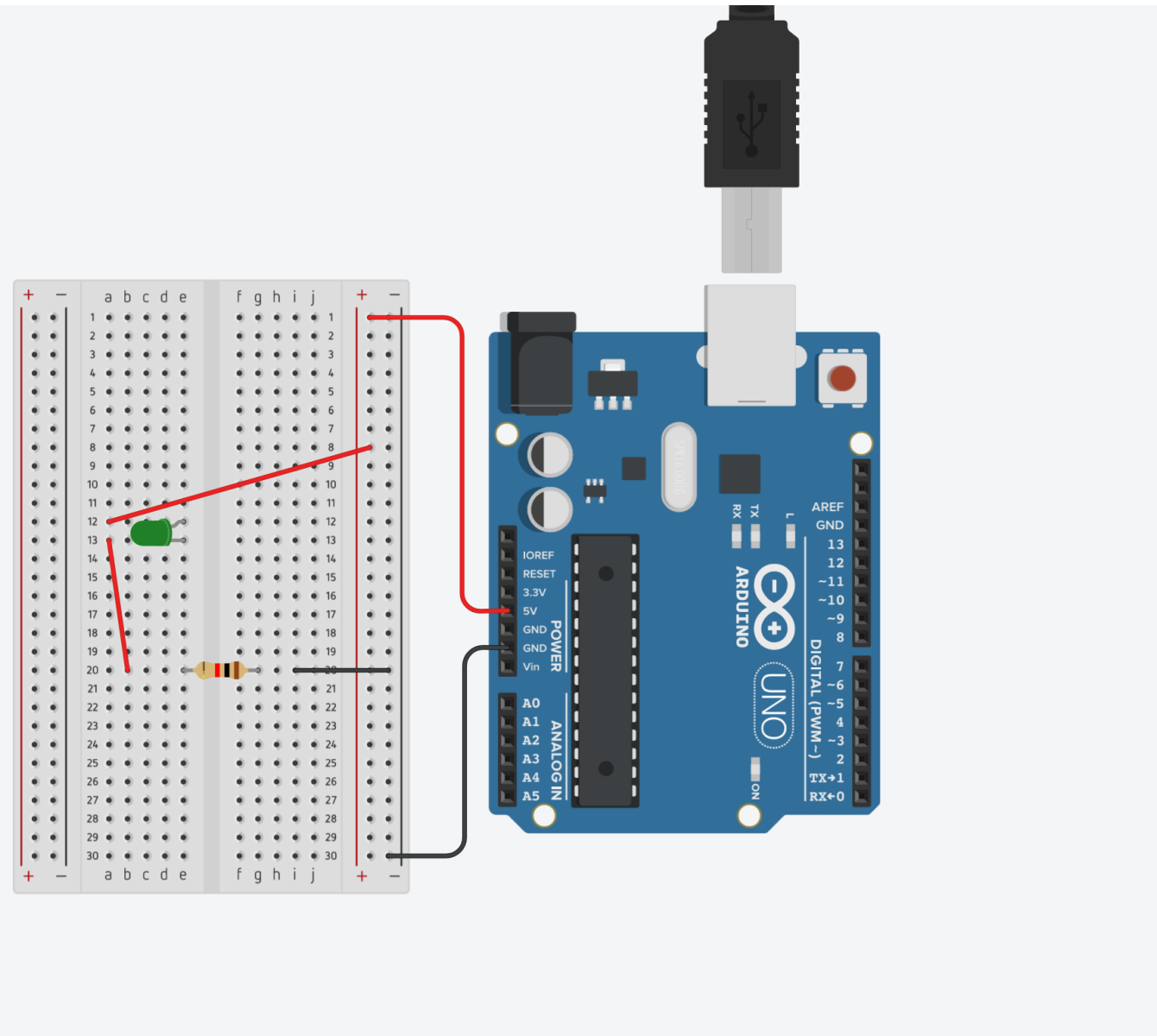
- Using your board, wires and components, try and recreate this circuit on the breadboard
- **Note: The + end of the LED is the LONGER leg**
- Remember: Power needs to continuously flow from + to -
 - 5V and GND pins respectively
- Also: the wire in the diagram follows a specific shape, you do not need to follow its specific path



Possible Solution (There are other ways, but what matters is that the circuit is complete)



Another Possible Solution



New Component: Button

- A button acts as a "bridge" electrically
- When not pressed the circuit line is broken
- When pressed the connection is completed and electricity can pass through
- The four legs can be plugged into breadboard slots
- There is no + or – end
 - + or – is determined by the circuit you build around it

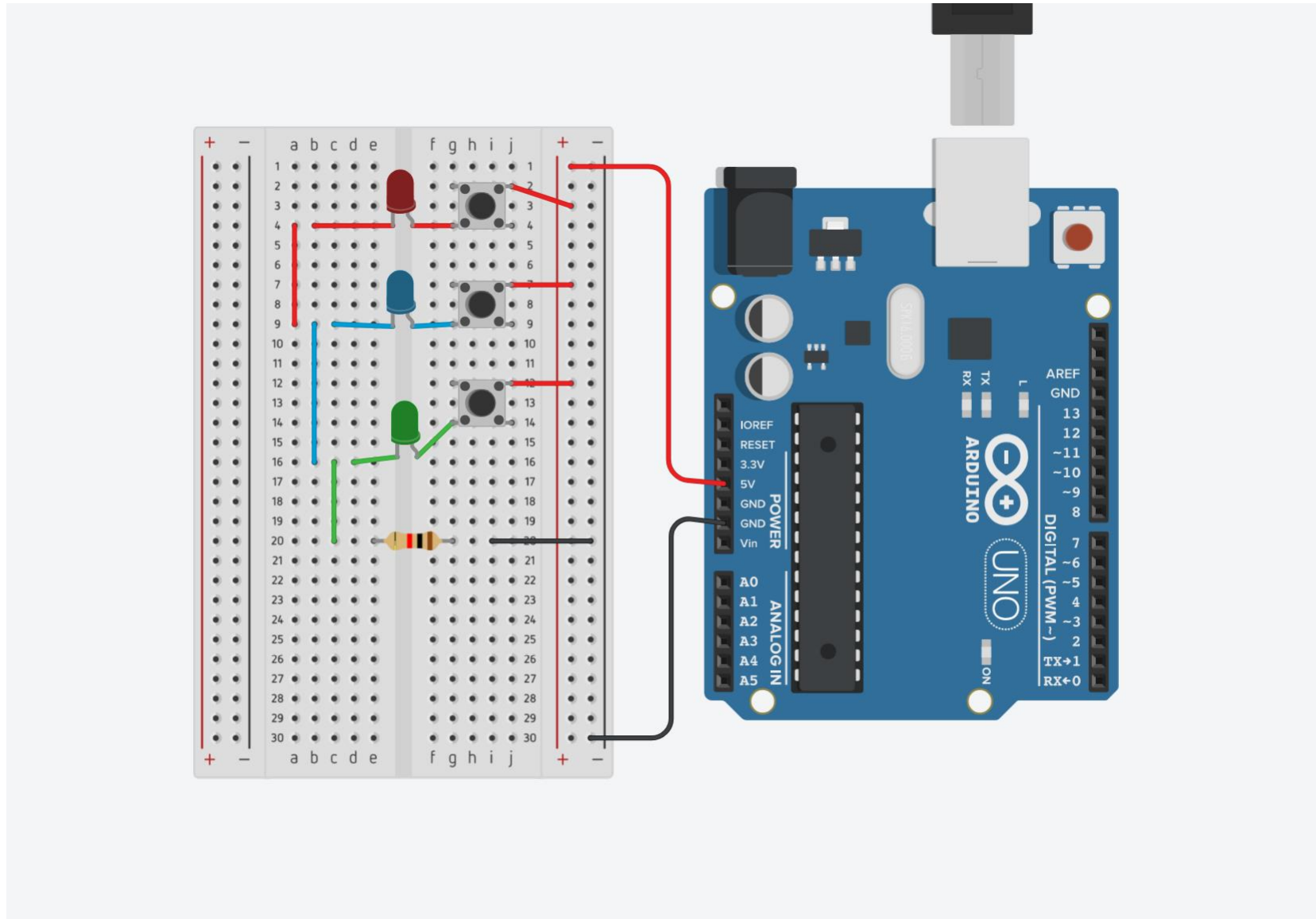


New Challenge: Button Mashing

- Create a similar circuit from before with:
 - 3 LED, 1 Resistor, 3 Buttons
- Make it so that when a button is pressed, one light turns on and another button for the second LED and so on
- Both lights must be connected to the same resistor
- Hint: Parallel vs series



Possible Solution:



Coding Time:



Getting Started: IDE

- IDE stands for "integrated development environment"
- It's basically where coding happens
- Fortunately, Arduino has one that makes it easy to connect to your boards and add cool add-ons
- It also features a place to download libraries and configure your code to whatever board you're using
- You can use other code editors, but this is the easiest to get started for arduino

Downloads



Arduino IDE 2.3.4

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

DOWNLOAD OPTIONS

Windows Win 10 and newer, 64 bits

Windows MSI Installer

Windows ZIP file

Linux AppImage 64 bits (X86-64)

Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.15: "Catalina" or newer, 64 bits

macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

[Release Notes](#)

Nightly Builds

Download a **preview of the incoming release** with the most updated features and bugfixes.

Coding Basics

- Coding is essentially providing instructions for your board
- It will interpret your instructions **literally** which means that most things are case sensitive
 - Ex: a variable named "value" is different than one name " value" (there is a space at the beginning)
- Everything in your code will happen in the order it is written; this can affect any logic you apply
- It is important to end each line with a ";" this denotes the end of each logic statement

```
void setup() { //just once!  
  Serial.begin(9600); // serial time!  
  lcd.begin(16, 2); //16 columns and 2 rows  
  Wire.begin();  
  pinMode(green, OUTPUT);  
  pinMode(red, OUTPUT);  
  pinMode(blue, OUTPUT);  
  if (mySensor.begin() == false) {  
    Serial.println("No SGP30 Detected. Check connections.");  
    while (1);  
  }  
  mySensor.initAirQuality(); //initialize sensor  
}  
void loop() { //repeats over again  
  //Serial.println(humipin);  
  //Default State is 400 ppm for first readings  
  delay(1000); //wait 1 seconds  
  int hum = DHT11.read(DHT11PIN);  
  lcd.setCursor(0,0); //sets cursor to home position  
  lcd.print("Hum "); //text  
  lcd.print((float)DHT11.humidity,0); // This *SHOULD* print  
  mySensor.measureAirQuality();
```

Coding Basics Continued

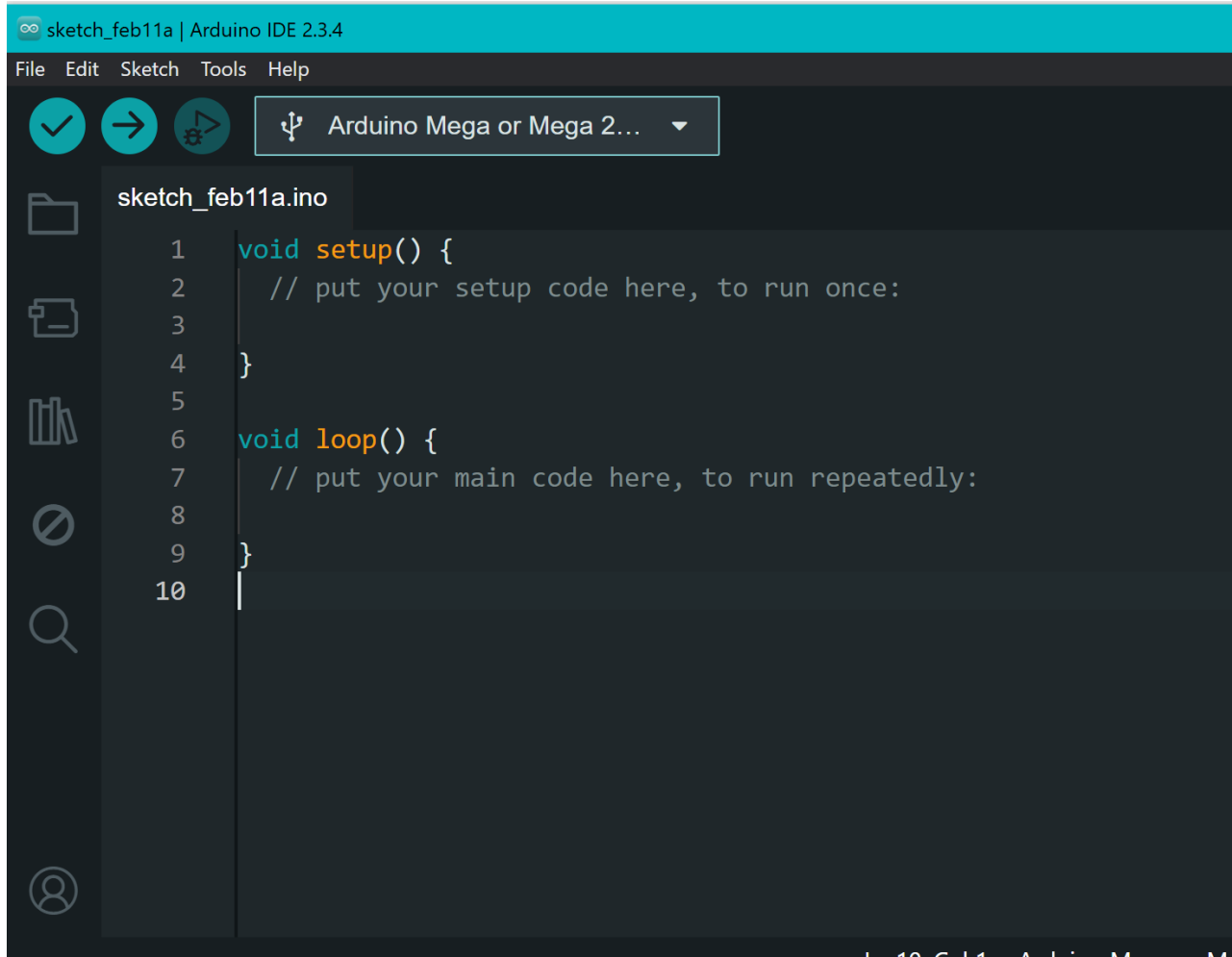
- **Variables** are "nicknamed" values that are what you define them as. You can choose if they change
 - Ex: `int ButtonCount = 2` means a number (int) named ButtonCount that is equal to 2
- A **function** is a sequence of instructions that begins when triggered
- **Comments** are written lines that have no effect on the code itself. They simply serve to explain to the reader what is going on
 - Always starts with a `/**`

```
void setup() { //just once!
  Serial.begin(9600); // serial time!
  lcd.begin(16, 2); //16 columns and 2 rows
  Wire.begin();
  pinMode(green, OUTPUT);
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT);
  if (mySensor.begin() == false) {
    Serial.println("No SGP30 Detected. Check connections.");
    while (1);
  }
  mySensor.initAirQuality(); //initialize sensor
}

void loop() { //repeats over again
  //Serial.println(humipin);
  //Default State is 400 ppm for first readings
  delay(1000); //wait 1 seconds
  int hum = DHT11.read(DHT11PIN);
  lcd.setCursor(0,0); //sets cursor to home position
  lcd.print("Hum "); //text
  lcd.print((float)DHT11.humidity,0); // This *SHOULD* print
  mySensor.measureAirQuality();
}
```

Basic Code Layout

- Arduino coding has three basic sections:
- **1:** Before anything in your document you can have a place to define any variables you want.
 - Remember that defining a variable is what brings it into existence, otherwise it does not exist!
- **2:** `void setup()` {code goes here}
 - This function will only run one time when the board turns on.
- **2:** `void loop()` {code goes here}
 - This function will repeat infinitely



```
sketch_feb11a | Arduino IDE 2.3.4
File Edit Sketch Tools Help
[Checkmark] [Next] [Upload] [USB] Arduino Mega or Mega 2...
sketch_feb11a.ino
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
10
```

Introducing Variables

- To introduce a new variable it needs to follow the following syntax:
 - [Optional modifier] [Type] [Name] = [Value]
 - Ex: `Const int ledPin = 10;`
 - This declares a **Constant Integer** named **ledPin** that is set to **10**
 - (this makes pin 10 be known as ledPin)
- **The constant modifier makes it so that the value never changes. Do not use this if you want the value to change!**

```
25 #include <LiquidCrystal.h>
26 #include <dht11.h>
27 #include "SparkFun_SGP30_Arduino_Library.h" // Click here to get the library: http://library
28 #include <Wire.h>
29 #define DHT11PIN 28
30 const int rs = 23, en = 22, d4 = 25, d5 = 24, d6 = 27, d7 = 26; //LCD Pins
31 const int red = 29;
32 const int blue= 31;
33 const int green= 30;
34 LiquidCrystal lcd(rs, en, d4, d5, d6, d7); //assign LCD Pins
35 dht11 DHT11;
36 SGP30 mySensor;
37
38
39 void setup() { //just once!
40     Serial.begin(9600); // serial time!
41     lcd.begin(16, 2); //16 columns and 2 rows
42     Wire.begin();
43     pinMode(green,OUTPUT);
44     pinMode(red,OUTPUT);
45     pinMode(blue,OUTPUT);
46     if (mySensor.begin() == false) {
47         Serial.println("No SGP30 Detected. Check connections.");
48         while (1);
49     }
50     mySensor.initAirQuality();//initialize sensor
51 }
52 void loop() { //repeats over again
53     //Serial.println(humipin);
54     //Default State is 400 ppm for first readings
55     delay(1000); //wait 1 seconds
56     int hum = DHT11.read(DHT11PIN);
```


PinModes

- Now that we know how to declare variables and pins, we can decide how the pins are going to behave
- You can do so in the setup() function
- Syntax: pinMode("pin variable name", [STATE])
- The state can be set to OUTPUT or INPUT
 - I'm not yelling, **uppercase does matter here**
- Output pins send out electricity (to power things)
- Inputs receive electricity and we can check if they have received data or just power

```
#include <LiquidCrystal.h>
#include <dht11.h>
#include "SparkFun_SGP30_Arduino_Library.h" // Click here to get the library: http://librarymanager.com#SparkFun+SGP30
#include <Wire.h>
#define DHT11PIN 28
const int rs = 23, en = 22, d4 = 25, d5 = 24, d6 = 27, d7 = 26; //LCD Pins
const int red = 29;
const int blue= 31;
const int green= 30;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7); //assign LCD Pins
dht11 DHT11;
SGP30 mySensor;

void setup() { //just once!
  Serial.begin(9600); // serial time!
  lcd.begin(16, 2); //16 columns and 2 rows
  Wire.begin();
  pinMode(green,OUTPUT);
  pinMode(red,OUTPUT);
  pinMode(blue,OUTPUT);
  if (mySensor.begin() == false) {
    Serial.println("No SGP30 Detected. Check connections.");
    while (1);
  }
  mySensor.initAirQuality();//initialize sensor
}

void loop() { //repeats over again
  //Serial.println(humipin);
  //Default State is 400 ppm for first readings
  delay(1000); //wait 1 seconds
  int hum = DHT11.read(DHT11PIN);
```

Activity: Blink

- In the files I provided go into Blink_example
- Take a quick second to read it and see how it works
- Breakdown:
- Void Setup
 - Sets the builtin pin (13) to emit electricity (output)
- Void Loop
 - Set pin 13 to high
 - Wait 1s (1000ms)
 - Set Pin 13 to low
 - Wait 1s
 - Repeat
- **Activity:** set up your led, resistor and breadboard to make the light blink! Make sure to plug in your board to the pc and press upload!!!



Activity: Blink pt 2

- Try and edit your code and wiring so now make it so that pin 12 has its own LED that will alternate with pin 13's blinking
- The sequence should be something like:
 - 13 on, 12 off
 - Wait
 - 12 on, 13 off
- Hint: you'll need to declare a new variable for the pin that's set to 12



Reading an input

- Now that we know how to do some basics lets try reading an input
- This can be done in the loop by using `digitalRead()`
 - This reads the pin for electricity or a signal
 - Ex: `int lightStatus = digitalRead(pin1);` (done before `setup()`)
 - This will make a variable that is equal to the value of what is in pin 1

```
void setup() {  
    // initialize the button pin as a input:  
    pinMode(buttonPin, INPUT);  
    // initialize the LED as an output:  
    pinMode(ledPin, OUTPUT);  
    // initialize serial communication:  
    Serial.begin(9600);  
}  
  
void loop() {  
    // read the pushbutton input pin:  
    buttonState = digitalRead(buttonPin);
```

Ok cool, but how do I actually *read* it?

- Introducing the ***serial monitor***
- In setup you can initialise the serial monitor by using `Serial.begin(9600);` in the `setup()`
- This opens a way for us to read variables and other data
- Can be read by opening the serial monitor at the top of the screen
- In this example `Serial.println(buttonState)` displays the button state in the serial monitor every loop
 - The `\n` at the end of `Serial.print` makes it add a new line every time

```
// the loop routine runs over and over again forever:
void loop() {
    // read the input pin:
    int buttonState = digitalRead(pushButton);
    // print out the state of the button:
    Serial.println(buttonState);
    delay(1); // delay in between reads for stability
}
```

Simple Logic Statements

- Sometimes you need to have things occur after a condition is met
- This can be done using "if" statements
 - Syntax: if (conditions/logic) { Do this code}
 - Conditions are simple checks in variables
 - Ex:
 - if (buttonstate == HIGH){
Serial.print("button pressed!");
delay(100);
}
 - This states that if the pin reading the button receives electricity, write "button pressed!" In the serial monitor
- After an if statement you can use other statements such as "else" to make a sequence of events for what happens if your condition is not met
- Also elseif can be used to denote other specific conditions

- Examples:
 - if (buttonState == HIGH){ //checks if button is pressed
DigitalWrite(ledPin, HIGH); //turns on light
}
else{ //if condition not met
DigitalWrite(ledPin,LOW); //Turns off light
}
- You can use many types of logic for your statements
- If X **is equal to** Y:
 - (x == y)
- If X **is not equal to** Y:
 - (x != y)
- If condition 1 **AND** condition 2:
 - (Condition 1 && Condition 2)
- If condition 1 **OR** Condition 2:
 - (Condition 1 || Condition 2)
- **Realistically ANY logical expression can be used!**

Final Activity

- Using one LED, Resistor and button make it so that:
 - Pin A leads to a button which then leads to Pin B
 - Using your code and logic statements, make it so that when the button is pressed: Pin C turns on a light
 - And when the button is released, the light turns off
- Hint: **Digital Pin** B needs to **read** Digital Pin A
 - **If** that happens, Pin C Needs to **output** voltage