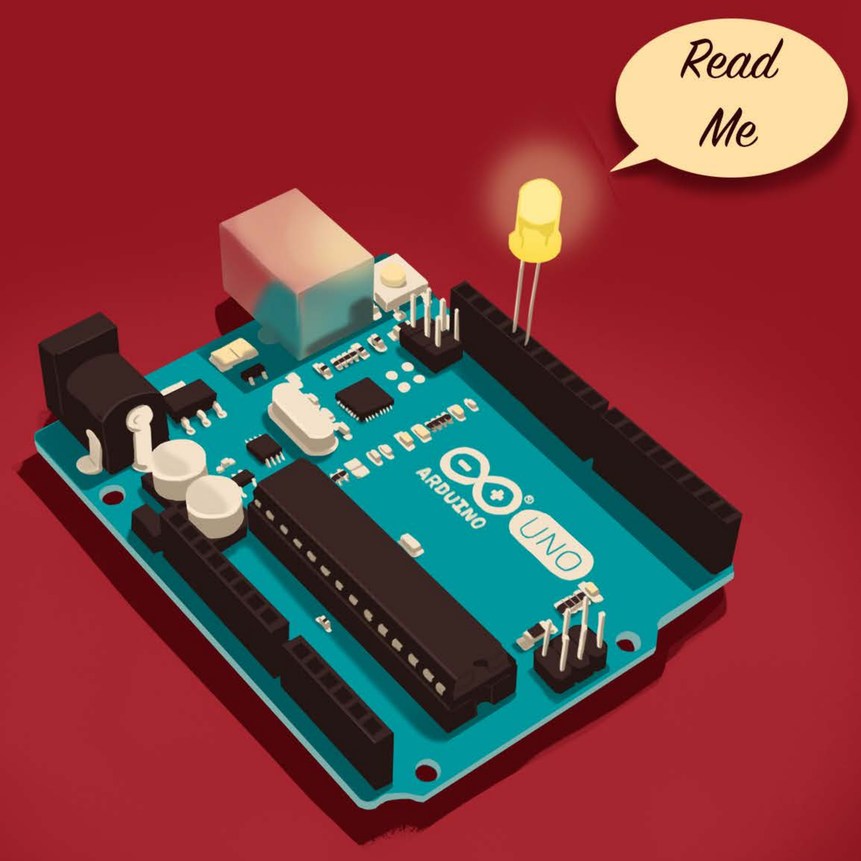
16Hz™ Presents

THE ILLUSTRATED ARDUINO



A USER-FRIENDLY GUIDE TO ELECTRONICS AND ARDUINO PROGRAMMING

The Illustrated Arduino, 1st Edition © 2015

Produced by: **16Hertz LLC**
1122 44th Rd
Long Island City, NY

Web: www.16Hertz.com
Email: support@16hertz.com
Social Media: @16HertzNYC

This guide is released under a Creative Commons Attribution, Non Commercial, ShareAlike 4.0 License.
Please share responsibly!

TABLE OF CONTENTS

Introduction 1

Tutorial 0 - Hello Circuits 5

Tutorial 1 - Digital Write 9

Tutorial 2 - Digital Read 16

Tutorial 3 - Analog Write 24

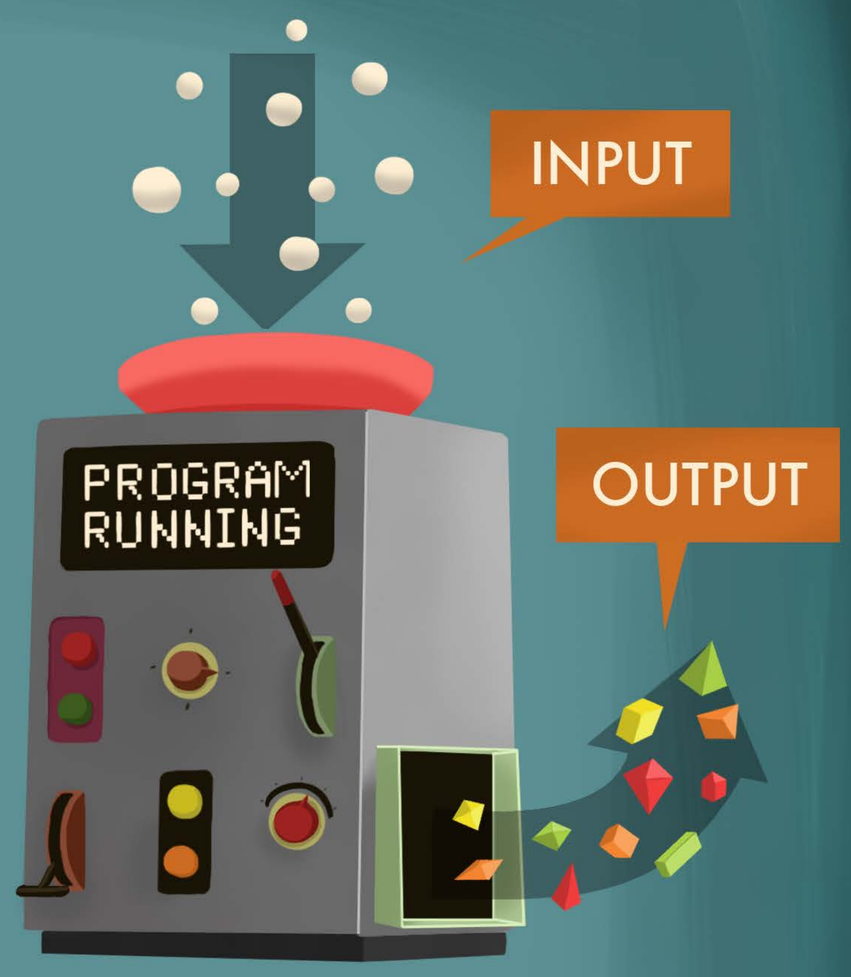
Tutorial 4 - Analog Read 39

Schematics & Diagrams 56

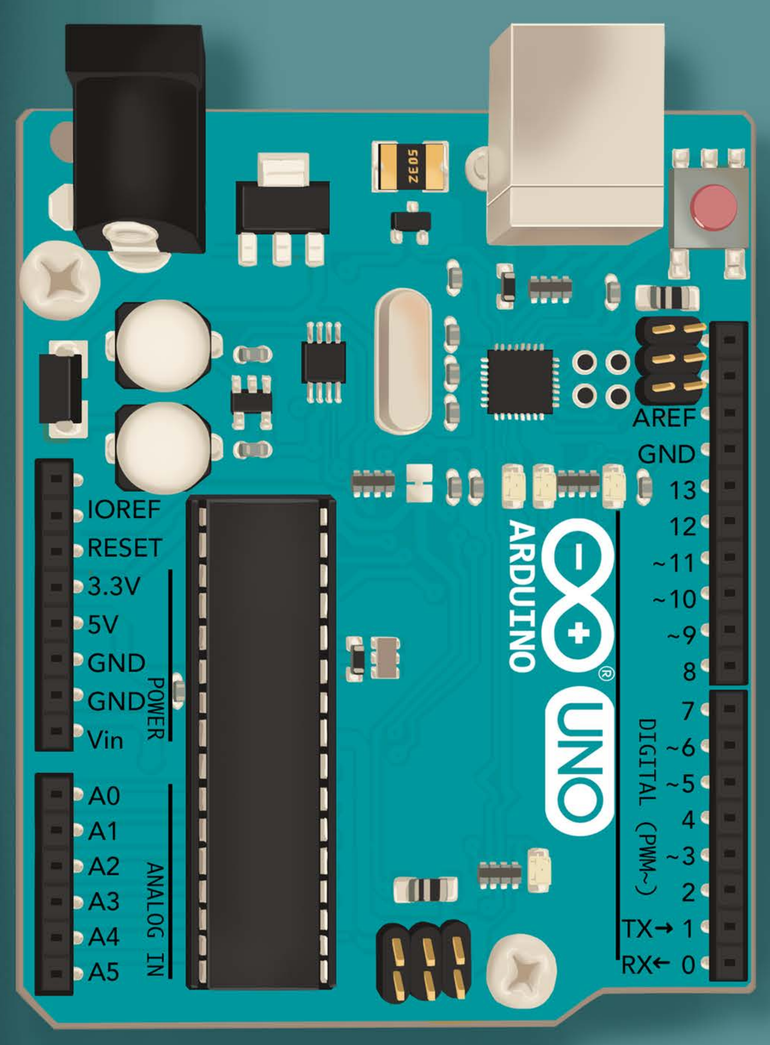
INTRODUCTION

What is Arduino?

Microcontrollers are at the heart of most electronic devices. They are essentially small computers, i.e. electronic decision makers that receive information as inputs and perform actions as outputs. INPUTs can be information from buttons and sensors while OUTPUTs are actions like blinking an LED or spinning motor. With a little programming, a microcontroller can be turned into a brain for controlling any electronics project, making it an excellent tool for rapidly prototyping ideas.



Arduinos are a series of open-source microcontrollers that were created by a team of Italian engineers. Since Arduinos are "open-source," anyone can adapt and improve upon their designs. It was this decision that allowed the Arduino community to thrive and revolutionized the world of hardware prototyping.



With of a line of versatile microcontrollers, an easy-to-use programming language, and a large community of makers and DIYers, the Arduino ecosystem is a perfect platform for electronics prototyping for anyone, regardless of experience level.

Arduino UNO, the board that this guide is built for, is the most widely used development board from the Arduino family of microcontrollers.

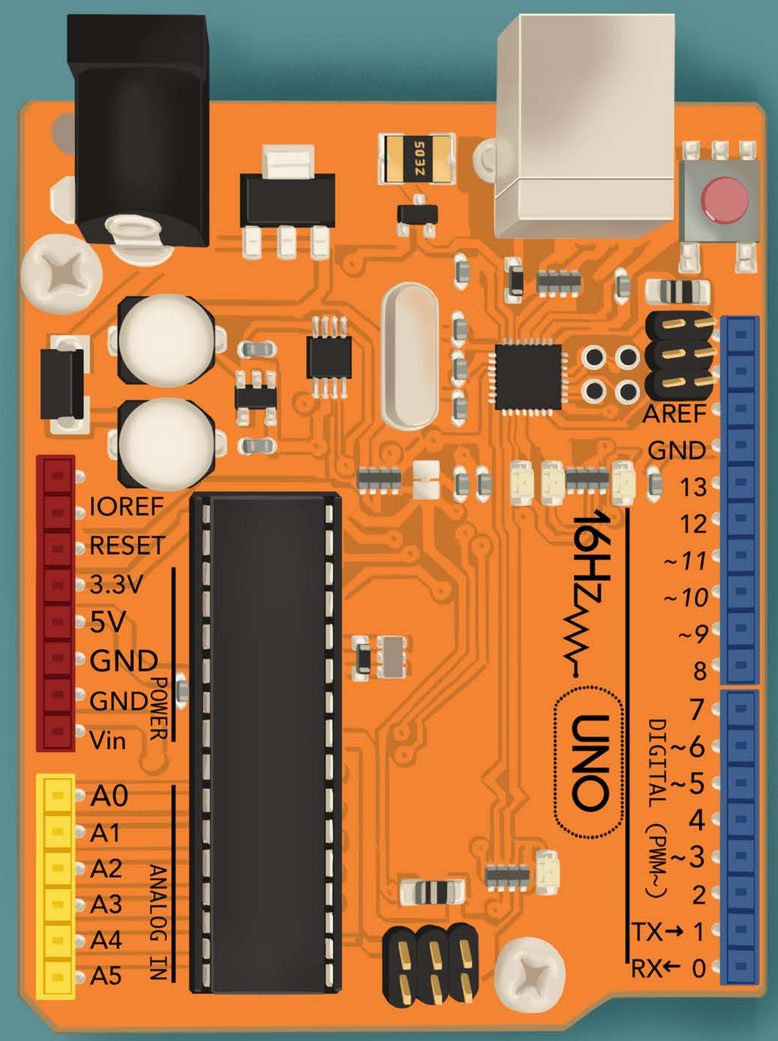
If any of this is unclear, don't worry! It will make a lot more sense later!

What Materials You'll Need

Hardware

The 16Hertz Ultimate Arduino Kit (www.16hertz.com/ArduinoKits) was specifically created to be a companion to this guide and has everything you need to get through all of the included projects. The kit includes an UNO R3 board with tons of other fundamental components. Though you can source all the individual parts separately, we recommend purchasing the 16Hertz Kit to minimize any compatibility issues (and cost!).

Note: You'll notice that the guide references the 16Hertz UNO board. This is our adapted version of the UNO, which is functionally identical to the Arduino version and substantially more affordable.



Software

In order to program your UNO board, you'll need to download and install the Arduino Integrated Development Environment, or IDE, from the Arduino website (www.arduino.cc/en/main/software). It is free to download and user-friendly, making uploading code to your UNO a snap.

If you're having any issues connecting to the UNO, be sure to follow the step-by-step guide on the Arduino website (www.arduino.cc/Guide).

Using The Guide

What You'll Get From It

This guide is intended to serve as an introduction to the world of electronics and microcontrollers for users of all experience levels. While a little background knowledge in electronics can be helpful, you can still get the most out of this guide with no experience at all.

How to Use It

While reading, you'll regularly find these helpful tools distributed throughout the guide:

Helpful Hints:

Helpful hints will appear in burgundy boxes and contain information you may find useful as you try to understand new concepts.

Pro-Tips:

Pro-Tips will give you information that, while not absolutely necessary, will help deepen your understanding of the content and/or provide some useful tips and tricks for the future.

Challenge:

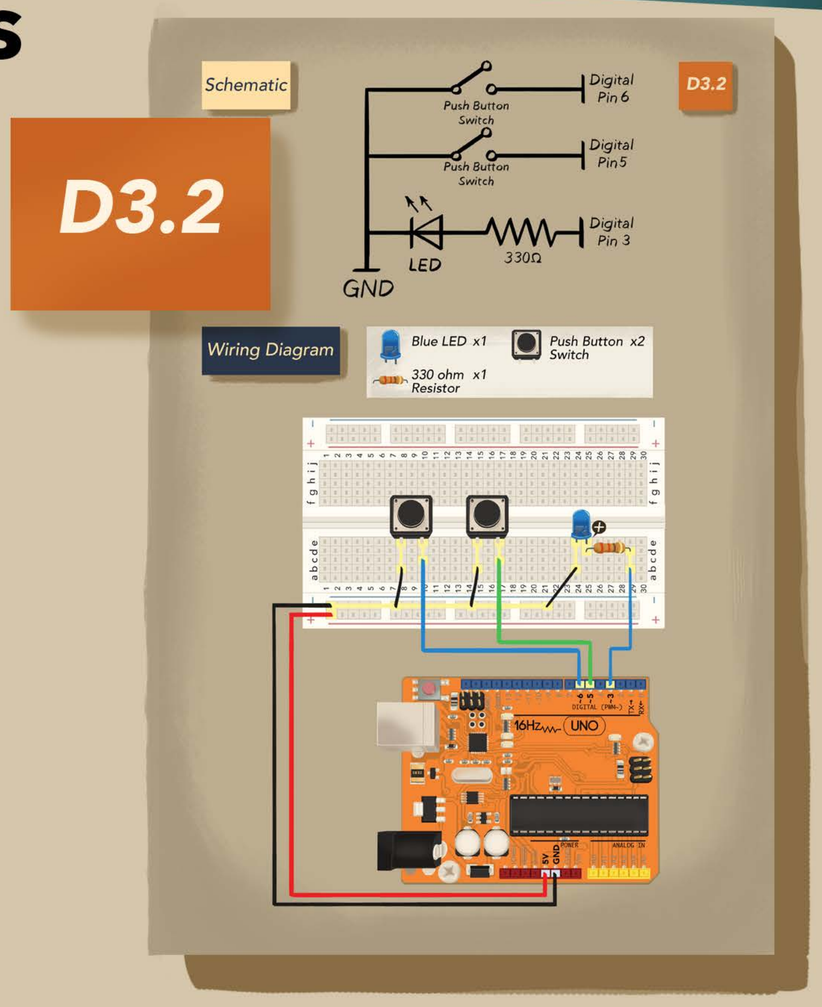
These sections will allow you to practice what you've read, give you an opportunity to hone your skills, and learn by problem solving. The more stars a challenge has, the harder it will be to solve.

Multimeter Moments

These icons will indicate when you should use your multimeter to measure various electronic properties of the current circuit. This will no doubt help you develop great debugging skills.

Wiring Diagram and Schematics

While you will be able to find all the wiring information you need within the tutorial pages, there are clearly labeled schematics and diagrams for each tutorial activity at the end of the guide. Just look for the beige label that tells you what part of the appendix you'll find it in. However, if you'd like to have a digital copy of all the code for all the tutorials in one place...



C1.2

```
void setup(){
      pinMode(3, OUTPUT);
}

void loop(){
      digitalWrite(3, HIGH);
      delay(1000);
      digitalWrite(3, LOW);
      delay(1000);
}
```

Code Panels

When you need to program your UNO, the code is clearly written inside the tutorials. However, if you'd like to have the code for all tutorials in one place, you can download them all at www.bit.ly/16hzcode. Just look for the orange label that indicates which program you'll need.

Let's Start Making!

We hope you have as much fun using this guide as we did developing it. Just remember, this is only the beginning. There is no limit to what you can eventually make with the skills you'll acquire!

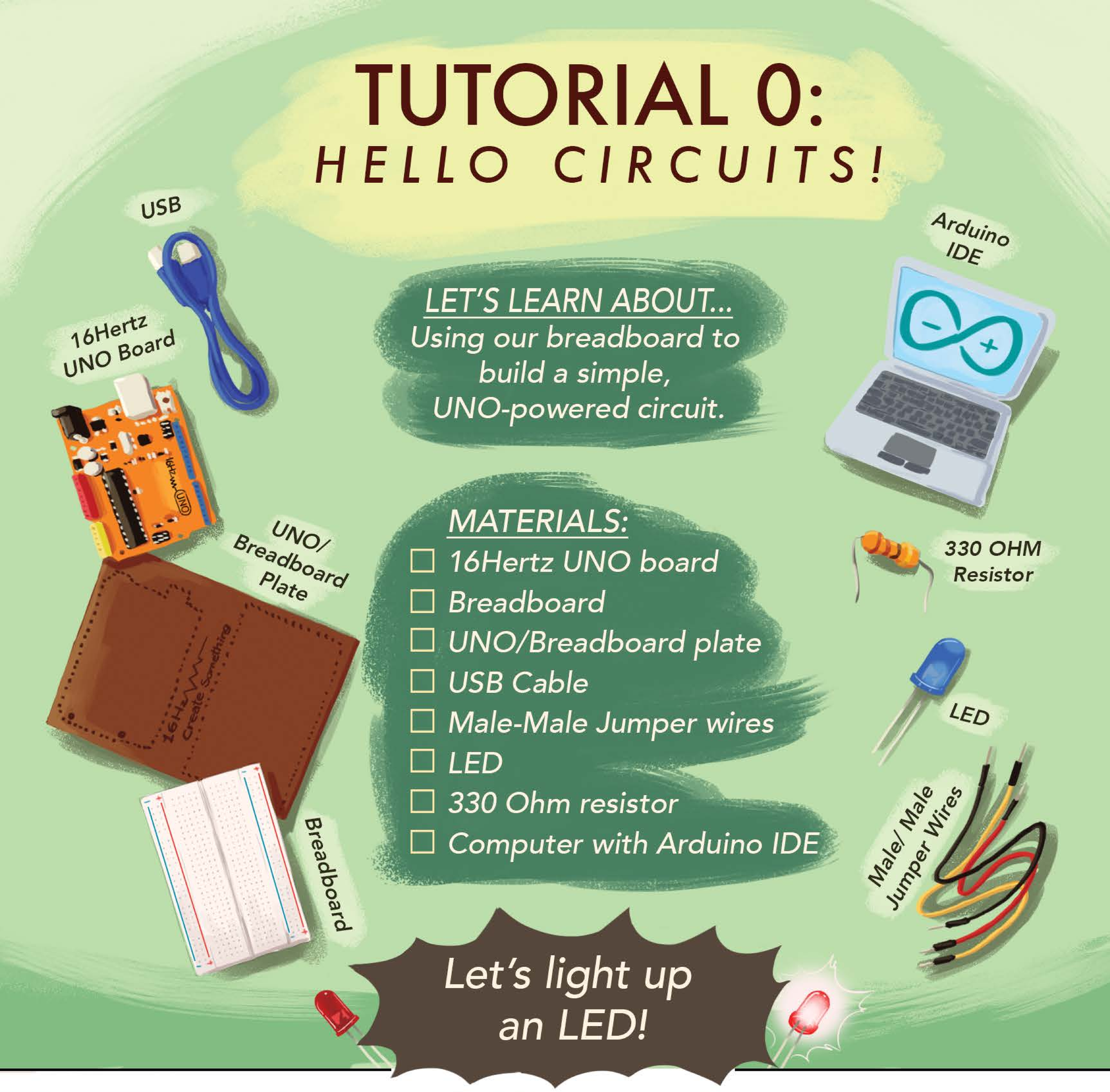
What you are looking at is an early stage of a more comprehensive electronics guide. While more tutorials, references, and resources will be added later, this current iteration will provide you with the basic skills you need to jump into your own creative Arduino projects.

If you'd like to provide feedback on how you think this guide can be improved, we'd love to hear from you! Please visit www.bit.ly/16hzfeedback to fill out a brief feedback survey.

MOHZ

CREATE SOMETHING

TUTORIAL 0:
HELLO CIRCUITS!

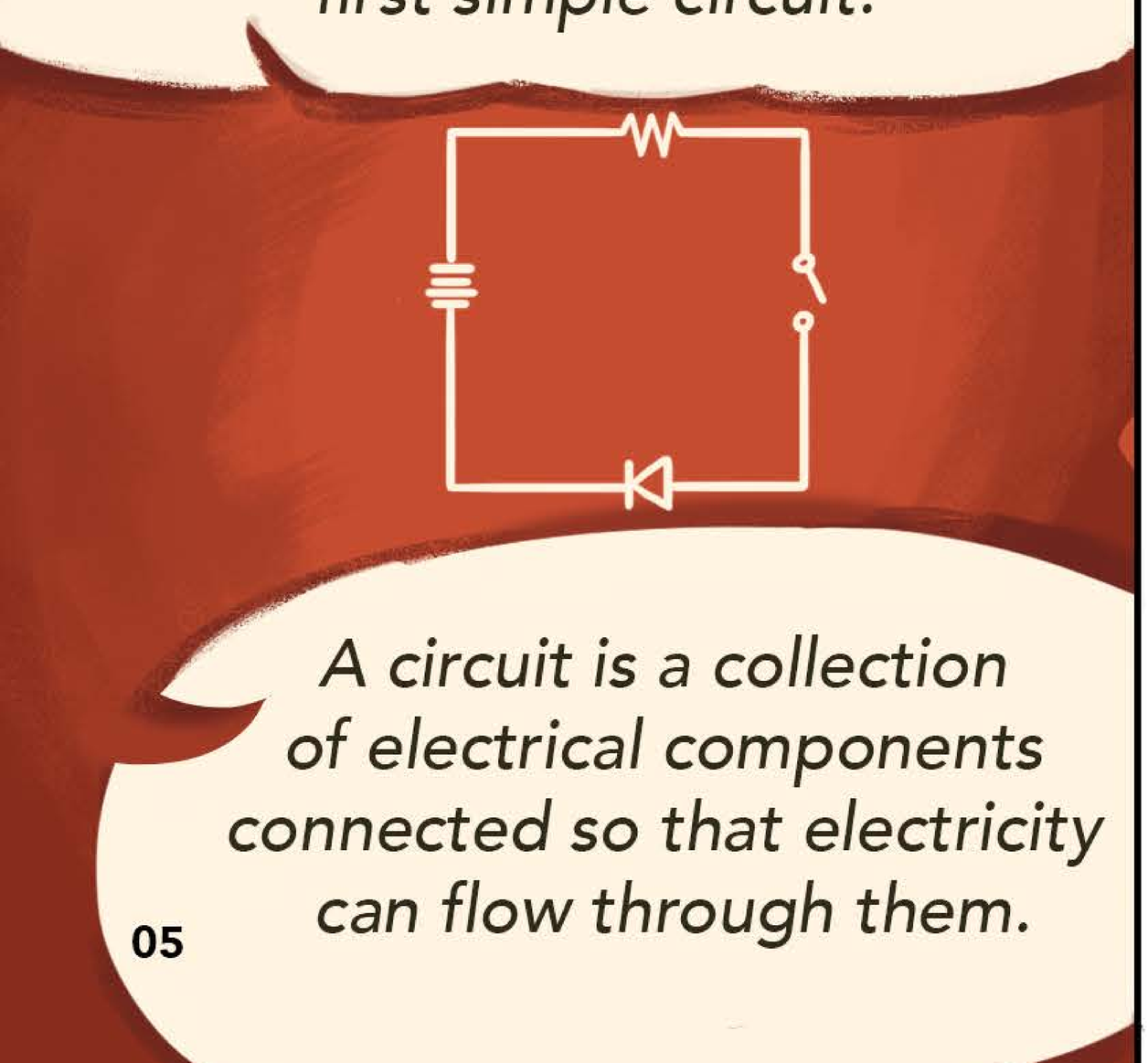


LET'S LEARN ABOUT...
Using our breadboard to build a simple, UNO-powered circuit.

MATERIALS:
- ☐ 16Hertz UNO board
- ☐ Breadboard
- ☐ UNO/Breadboard plate
- ☐ USB Cable
- ☐ Male-Male Jumper wires
- ☐ LED
- ☐ 330 Ohm resistor
- ☐ Computer with Arduino IDE

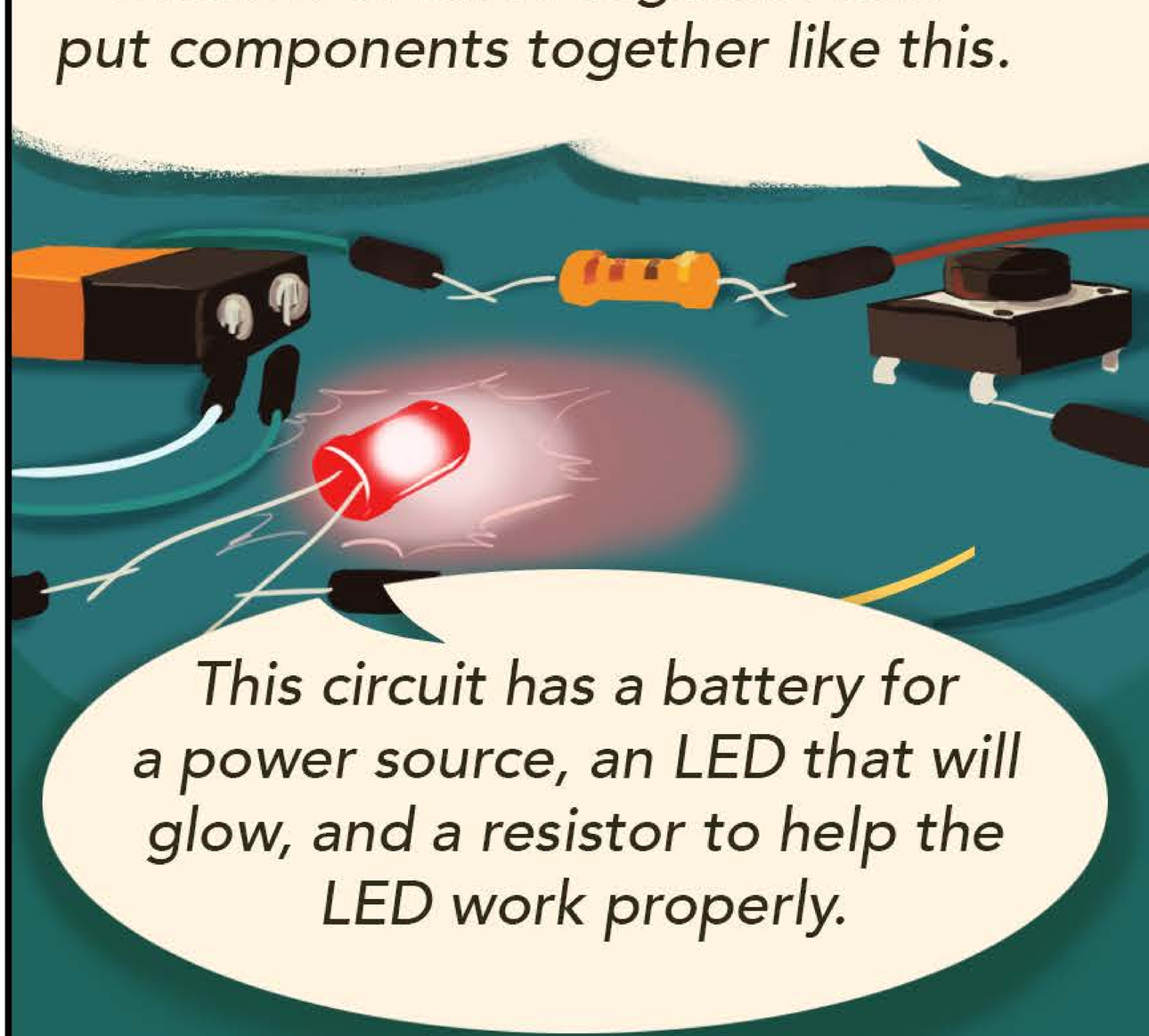
Let's light up an LED!

To do this, we'll build our first simple circuit.



A circuit is a collection of electrical components connected so that electricity can flow through them.

One way to make a circuit is to hold a bunch of wires together and put components together like this.

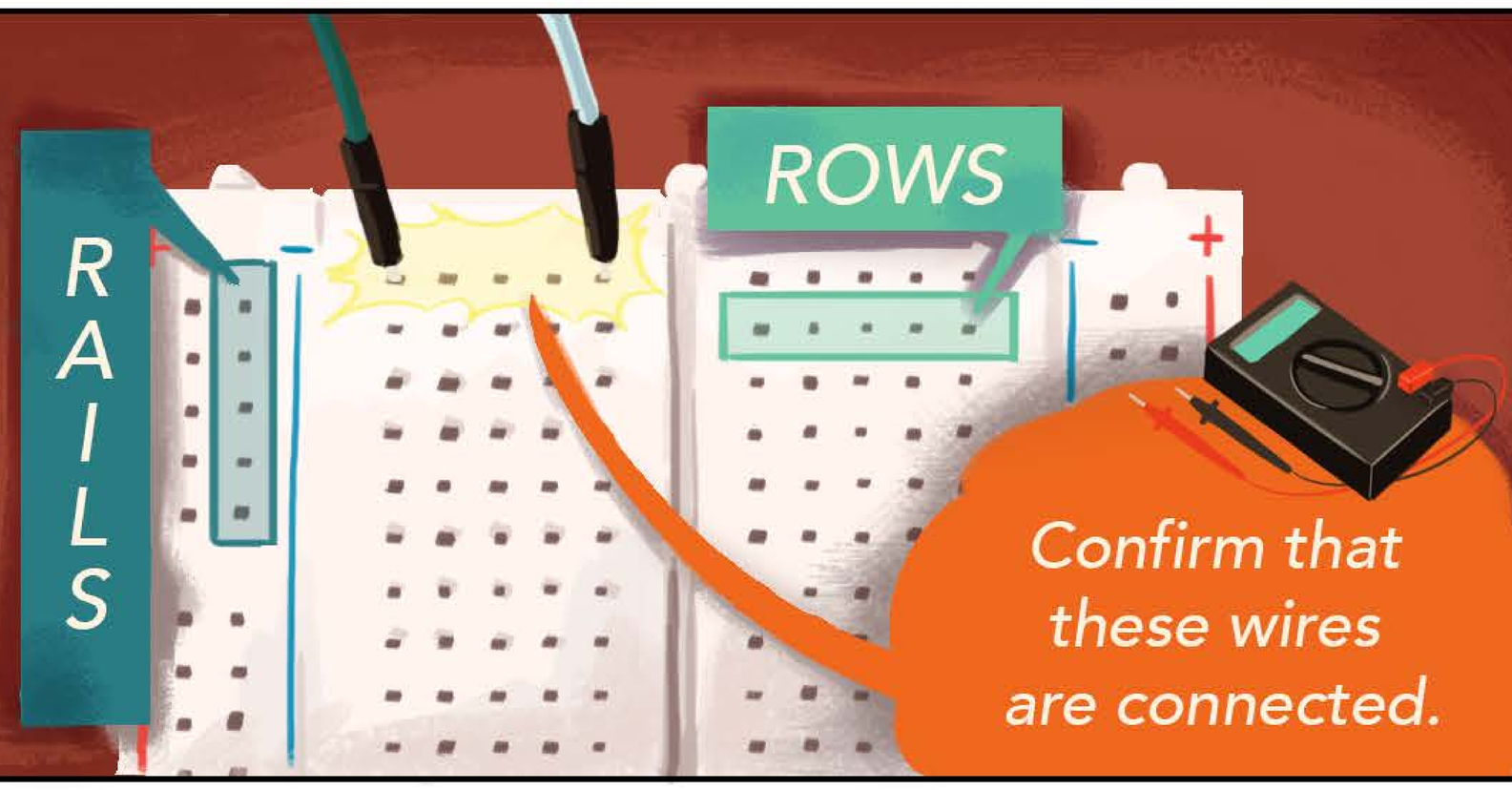


This circuit has a battery for a power source, an LED that will glow, and a resistor to help the LED work properly.

Instead of holding the components together, we'll use a "breadboard", a handy device that lets us make quick and reliable connections.

Strips of metal connect the holes in different patterns.

The metal strips inside the breadboard allow us to make a connection between two wires by simply plugging them into adjacent holes. How convenient!



The first thing we'll do is give our breadboard access to a power supply.

The UNO board gets its power from the computer.

Instead of using a battery, let's use the 16Hertz UNO board to give us a nice steady supply of power.

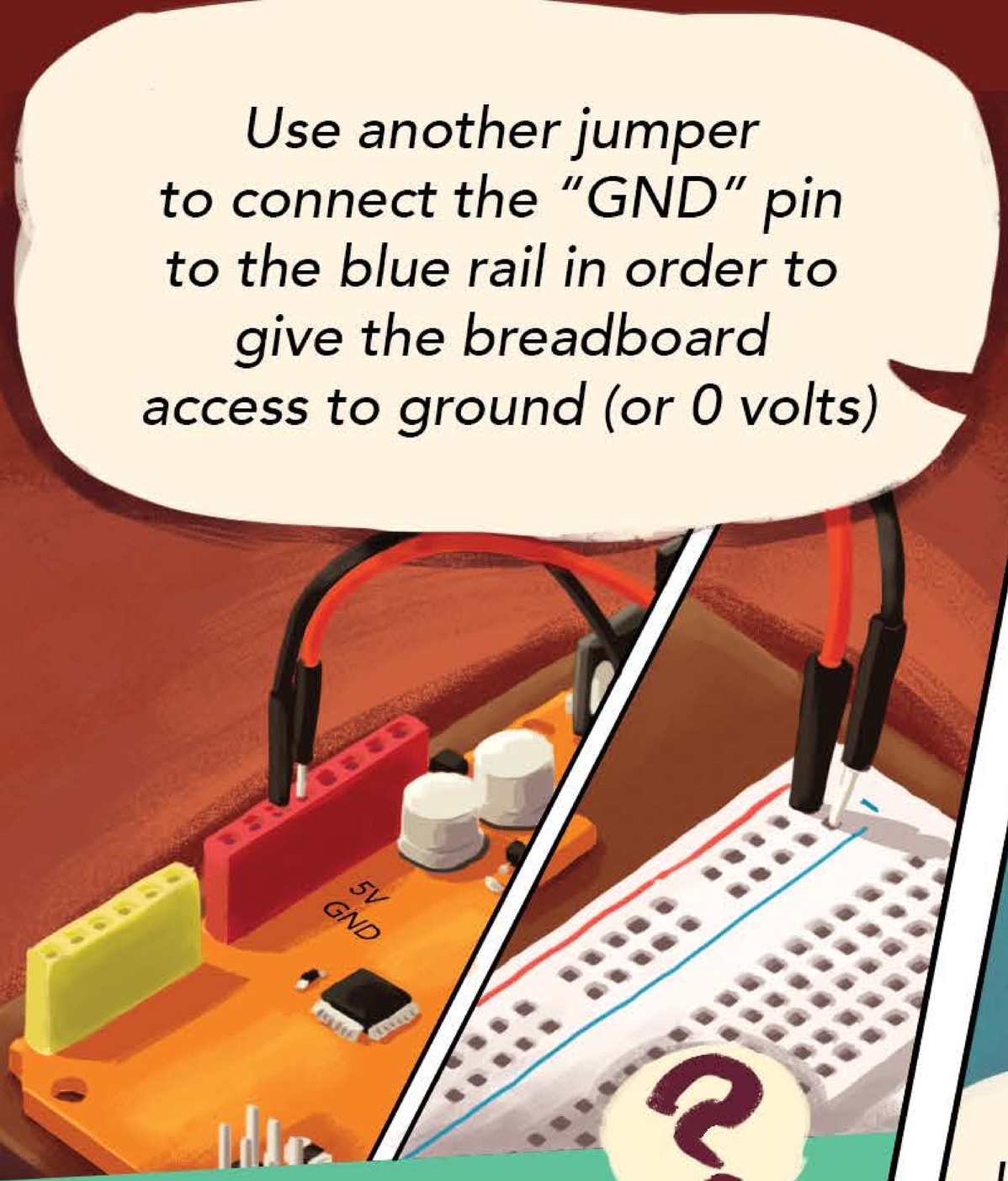
Start by plugging in a jumper wire from the "5V" pin on the UNO to the red "rail" on your breadboard.

Now all the holes on the red rail are connected to 5 volts.

Helpful Hint:

Jumper wire colors don't matter, but use the colors to keep track of them and stop your project from looking like rainbow spaghetti!

Use another jumper to connect the "GND" pin to the blue rail in order to give the breadboard access to ground (or 0 volts)



Reference:

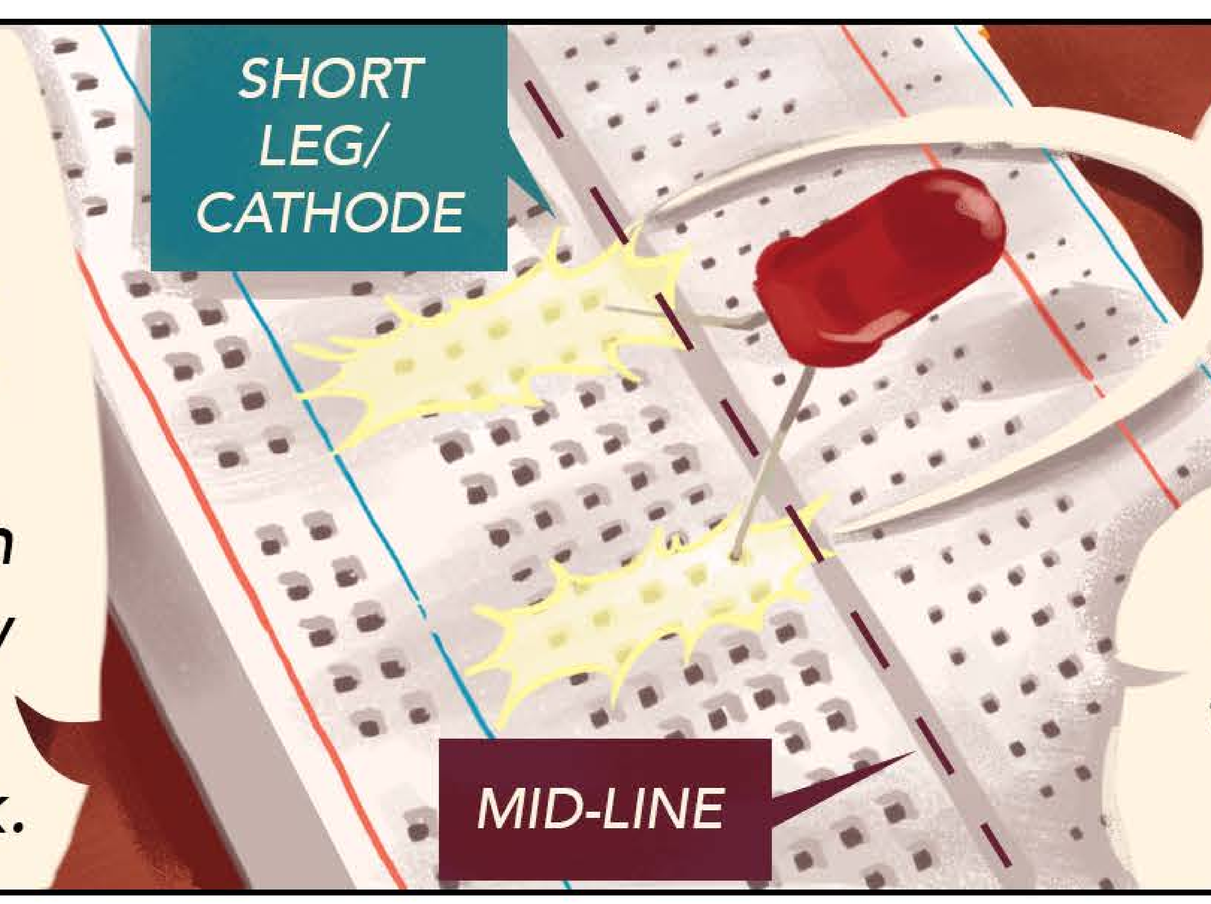
To find out more about voltage, read the voltage section in Appendix A.

Now let's do something with this voltage. Grab an LED and take a close look at the bulb and two legs.

The short leg/ flat side of the bulb is the "cathode" and should be connected to ground.

The long leg or "anode" is connected towards the positive voltage.

LEDs are "polarized" components, which means they have to be plugged in a specific way in order for them to work.



Plug in the LED's legs in different rows.

Rows are connected horizontally, but not to rows on the other side of the mid-line.

Now let's give the LED power so it can light up... But don't do it just yet!

If it gets too much current, it can burn out or even blow up!

To limit this current, we'll use a 330 OHM resistor.

Reference:

For more information on current and resistors, read about them in Appendix A.

Be sure to use the one with the orange-orange-brown band pattern.

Plug the resistor into the breadboard with one leg in the row with the LED's anode (the long leg).

By connecting the resistor this way, the LED will get less current so it doesn't burn out.

The LED and resistor are ready to be powered up! connect them to power and ground using two jumper wires.

Plug one into the 5 volt rail and the resistor's free leg. plug another into the ground rail and the led's cathode.

PRO-TIP:

The order of the LED and resistor do not matter as long as they are both between 5 volts and ground and the led's cathode is toward GND. The same current will flow through both components regardless of which one goes first.

In this simple circuit, the current flows from the 5 volt pin on the UNO, into the breadboard, where it goes through the resistor, the led, and then into ground.

Closed circuit = current flowing

Open circuit = No current flow

Because all connections are secure and the current is flowing, this is a "closed"circuit.

If you unplug any of the wires, the circuit would be "open" and no current would flow.

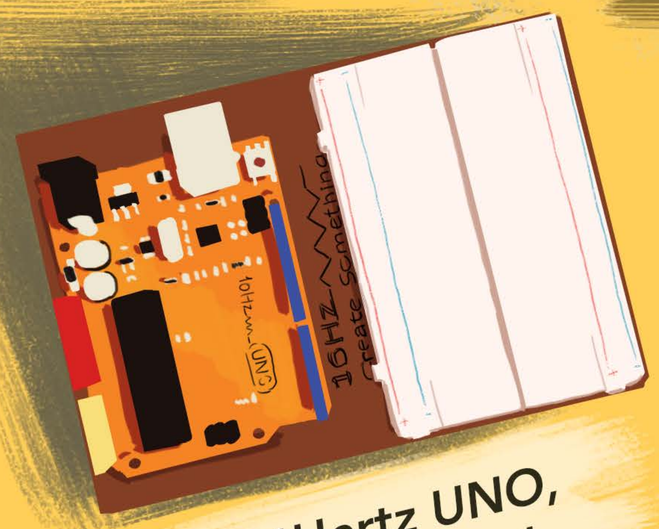
Try making the LED blink by opening and closing the circuit repeatedly. This is definitely the most annoying way to make an led blink. In the next tutorial, we'll use some code to make the UNO blink the LED for us!

Keep your circuit in one piece, you'll use it in the next tutorial!

TUTORIAL 1:
DIGITAL WRITE

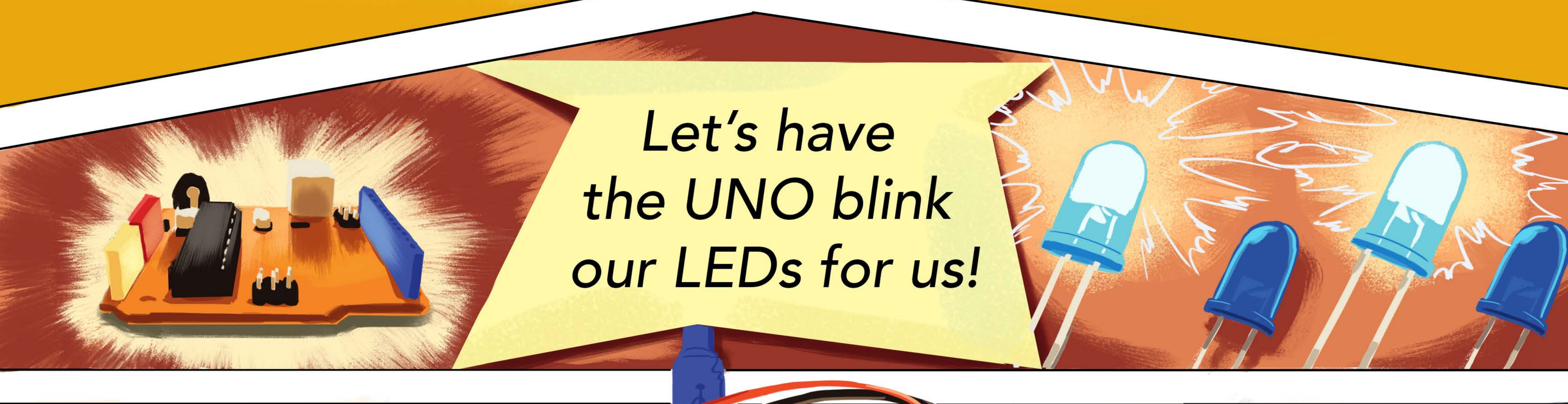
LET'S LEARN ABOUT...

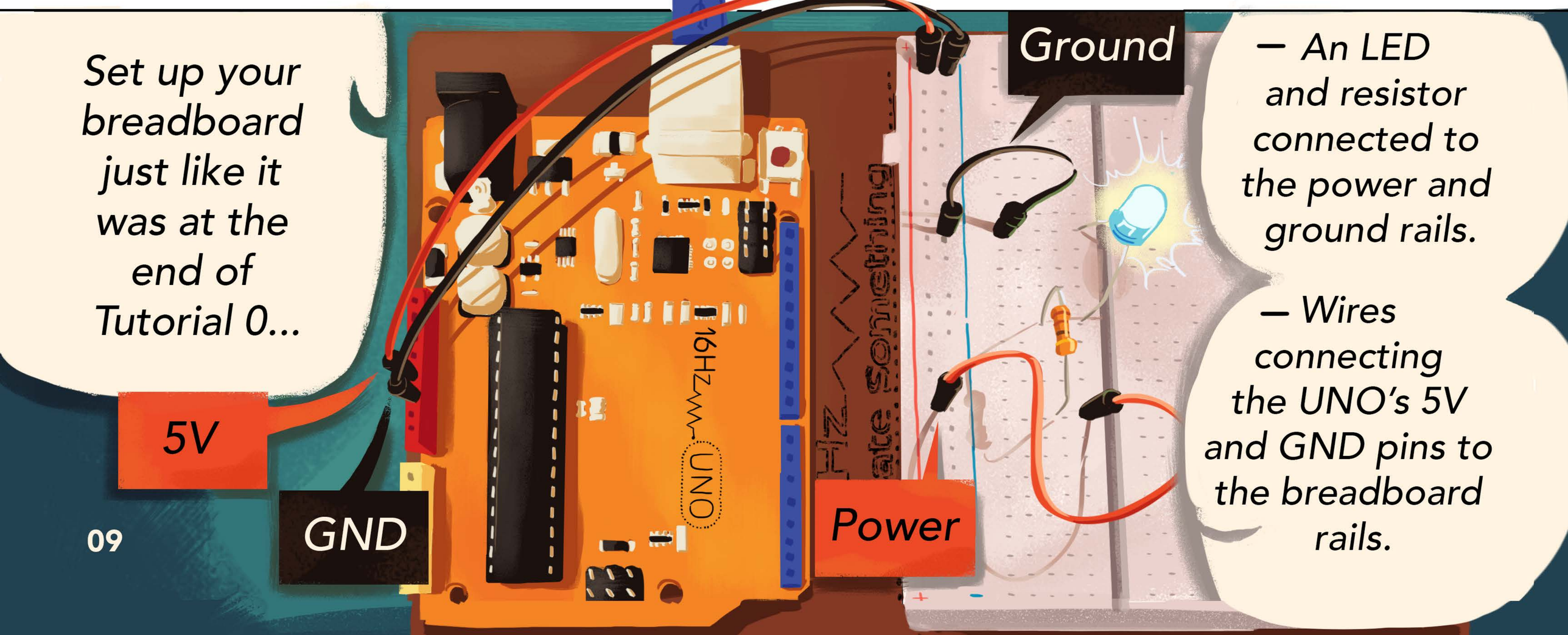
Using the `digitalWrite()` command to control the UNO's digital pins



MATERIALS:

- ☐ 16Hertz UNO board
- ☐ UNO/Breadboard plate
- ☐ Breadboard
- ☐ USB cable
- ☐ Male-Male Jumper wires
- ☐ LEDs
- ☐ 330 Ohm Resistors
- ☐ Computer & Arduino IDE download at http://www.arduino.cc





When connected to the 5 volt rail, the LED will always be lit unless unplugged. Now we're going to tell the UNO to turn it on and off using a **digital pin**. These are pins that can switch between Zero and 5 volts on command.

HEY PIN 3, BE 5 VOLTS! PIN 7, YOU BE 0 VOLTS!

Digital Pins

YES SIR!

Take the jumper wire that connects the LED to power and move it from the 5 volt rail to **digital pin 3** on the UNO.

Now when we tell pin 3 to be 5 volts, current will flow through the LED making it light up.

HI, I'M 5 VOLTS NOW. HERE IS SOME CURRENT.

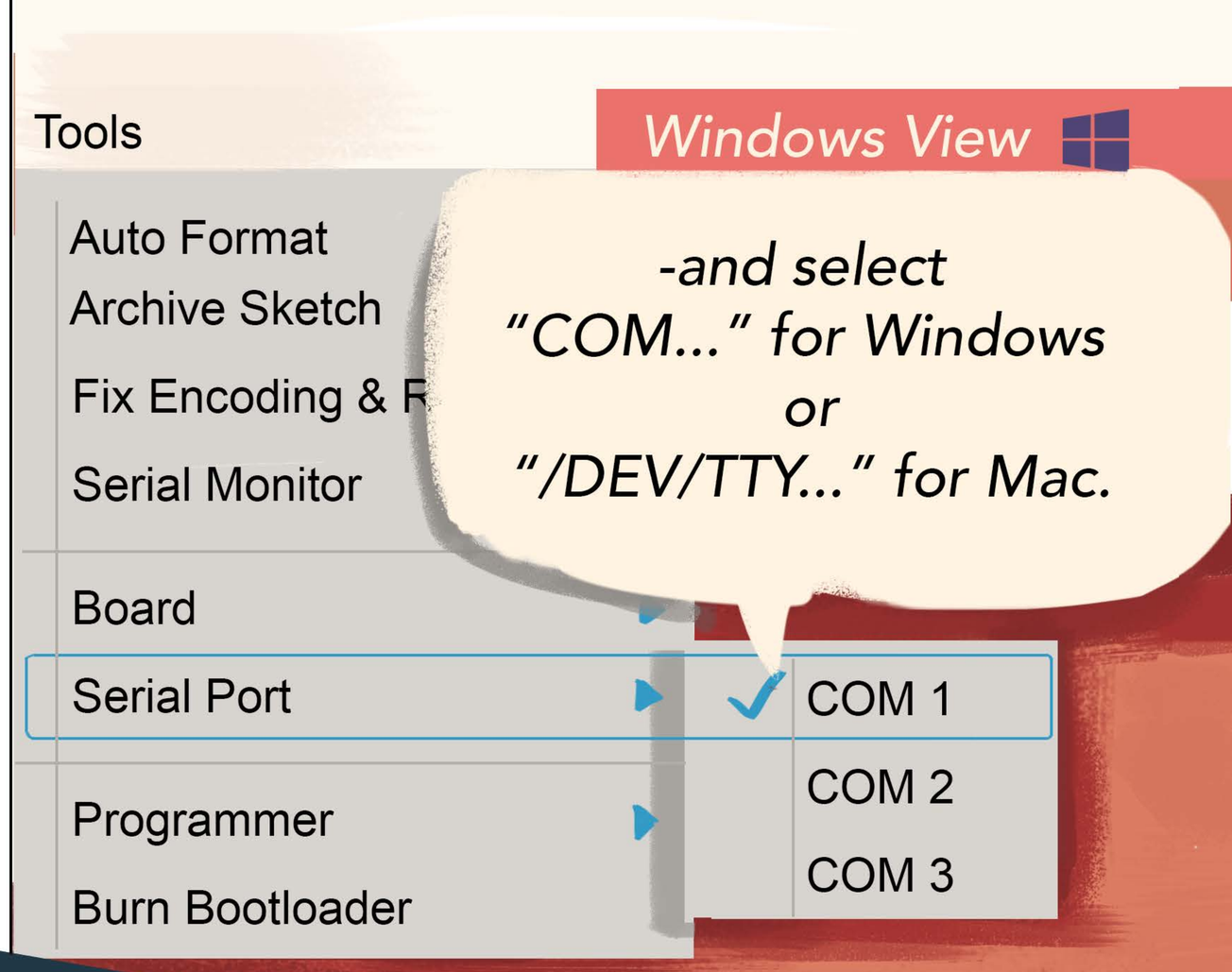
Pin 3

D1.1

DO IT.

That's all the hardware we need for now! To tell the UNO how to behave, we'll need the Arduino Integrated Development Environment, or IDE. Open up your computer and start the program so we can upload code to the UNO.

OK!



Pro-Tip:

If you have multiple options in the "Serial Port" menu, look at the menu before you plug in your UNO and again after. The new option will be your UNO. Pick that one.

The Arduino IDE has a lot of helpful features built in for you to explore, but here are some good things to know right off the bat.

```
Main Window:
//This is where your code goes.
```

Verify — *Use this to make sure your code is correct.*

Upload — *Use this to upload your sketch to the UNO.*

New/Open/Save — *For managing all your sketches.*

Serial Monitor — *Opens a window that listens for your UNO's messages.*

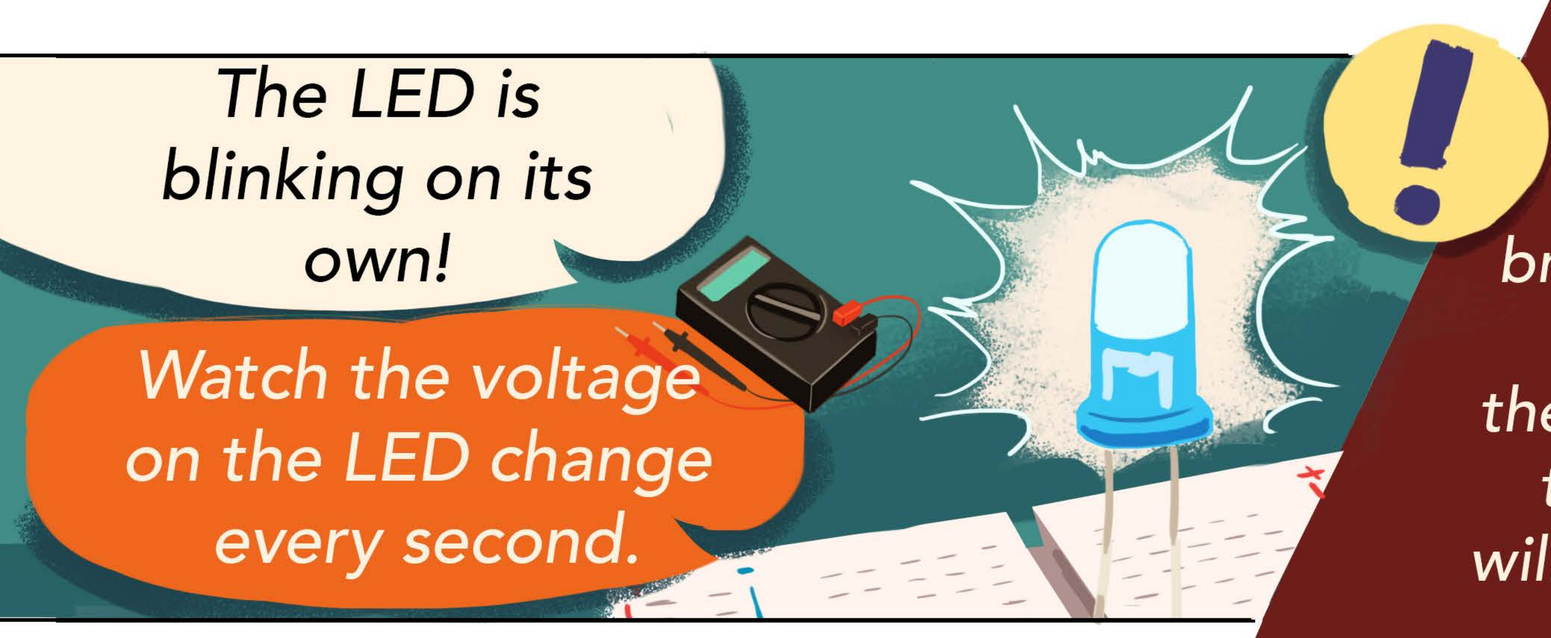
Console: This is where status and error messages appear.

```
Arduino: 1.0
core.a(main.
/Applications/Arduino.app/Contents/Resources/Java/hardware/arduino/cores/arduino/main.cpp:40: undefined reference to `setup'
/Applications/Arduino.app/Contents/Resources/Java/hardware/arduino/cores/arduino/main.cpp:43: undefined reference to `loop'
```

Type out this code and upload it to your UNO by pressing the Upload button. Don't forget the semicolon at the end of each line. Also, spaces don't matter, but capitals do.

C1.1

```
void setup(){
    pinMode(3, OUTPUT);
}

void loop(){
    digitalWrite(3, HIGH);
    delay(1000);
    digitalWrite(3, LOW);
    delay(1000);
}
```

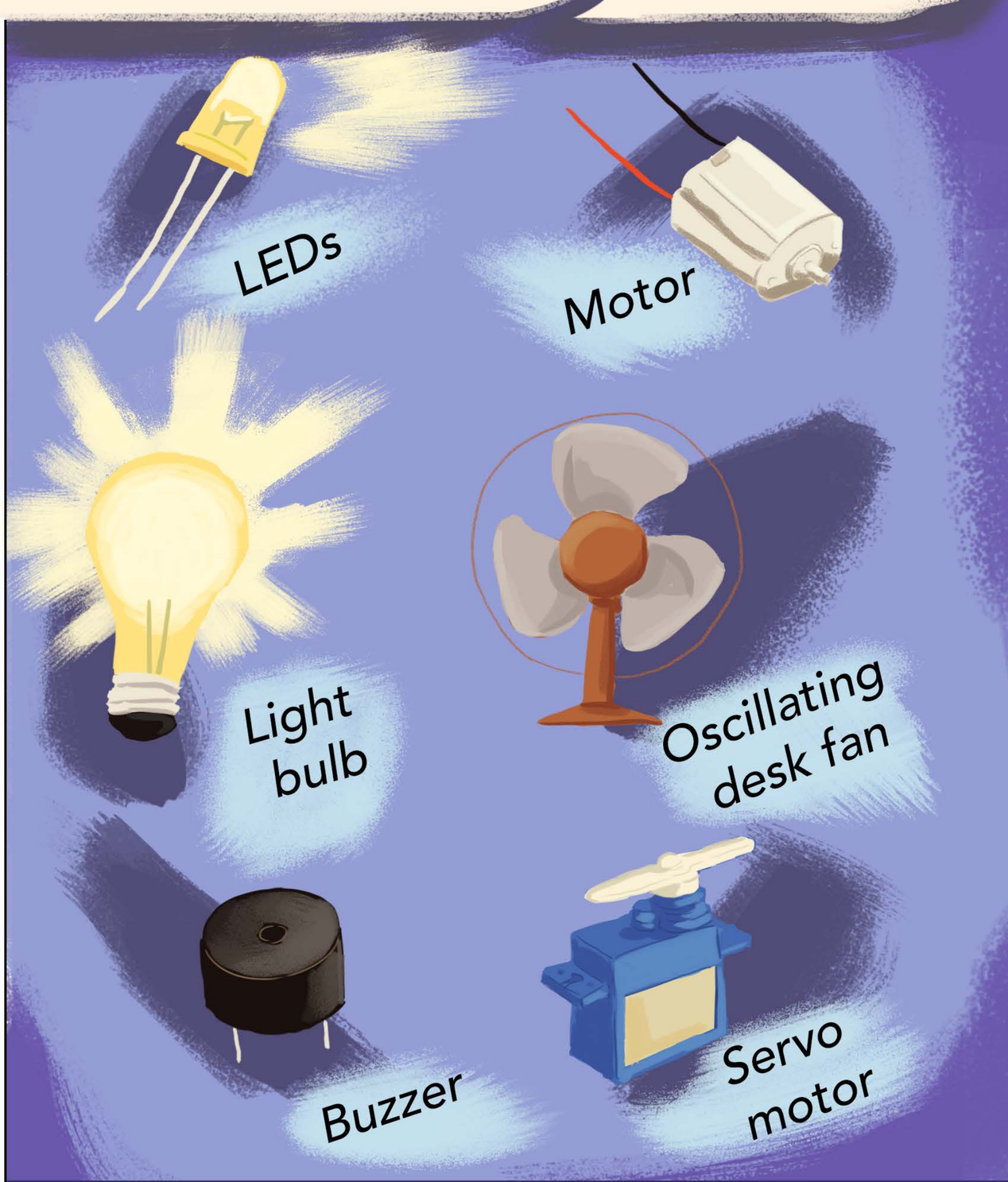


Helpful Hint:

All the code within the curly braces "{...}" after `void setup()` *happens only ONCE when the UNO starts. Everything within the* `void loop()` *curly braces will happen over and over, forever.*

Digital pins on the UNO can be either INPUTs or OUTPUTs, so we used the code `pinMode(3, OUTPUT);` *to make pin 3 an* `OUTPUT`.

INPUTS...
get information from the outside world.



OUTPUTS...
are things that affect the outside world.

Since the pin is now an OUTPUT, we can use the digitalWrite() command to tell pin 3 to be HIGH or LOW (on or off, respectively).

The first number sets which Pin we want to control.

Don't forget these!

```
digitalWrite(3, HIGH);
```

Note how the 'W' is capitalized.

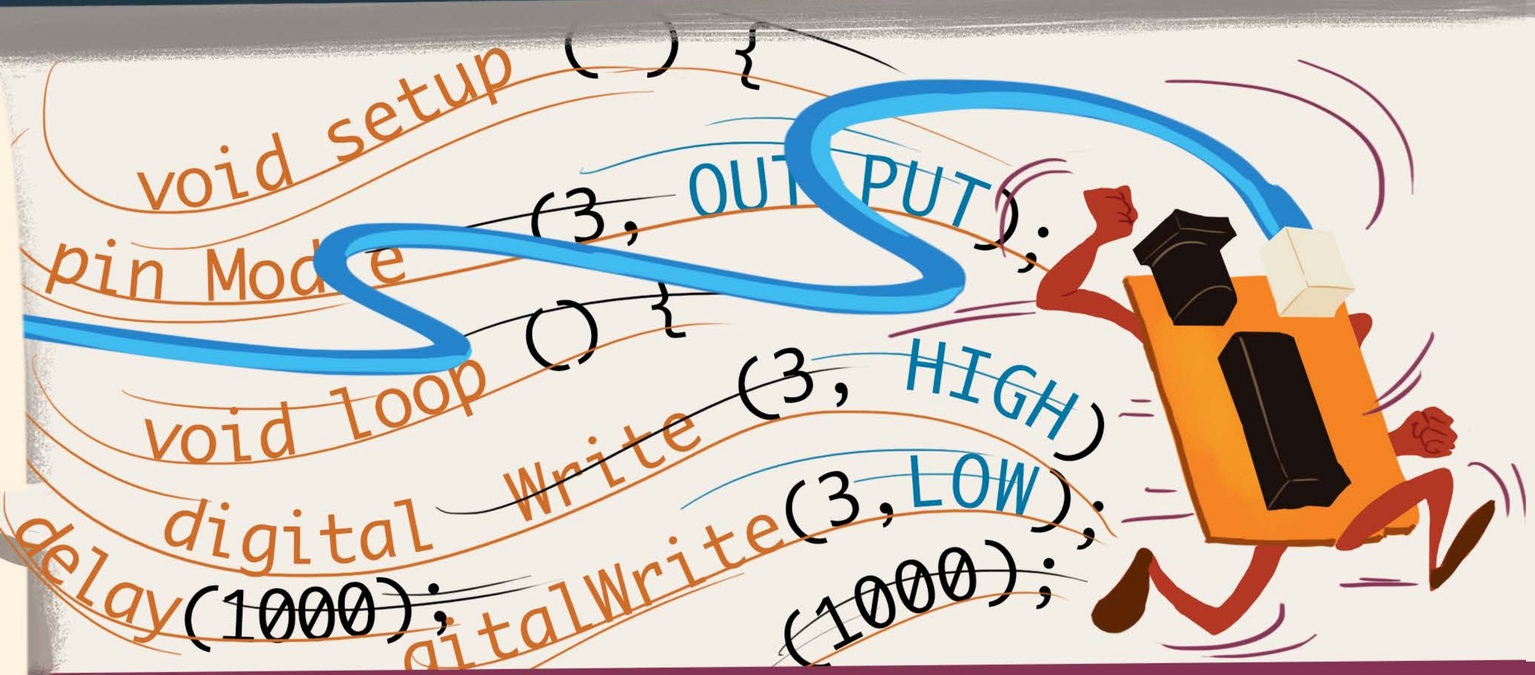
The second value in the parenthesis sets the pin to HIGH or LOW.

Pin 3 is now 5 volts, which makes the LED light up. This pin will stay on until we tell it to be off. The delay(1000) command tells the UNO to wait for 1000 milliseconds (1 second) before going to the next line.

Pro-Tip:

The delay() function will freeze everything on your UNO for the specified amount of time before moving to the next line. Be careful when using it because it could cause other problems.

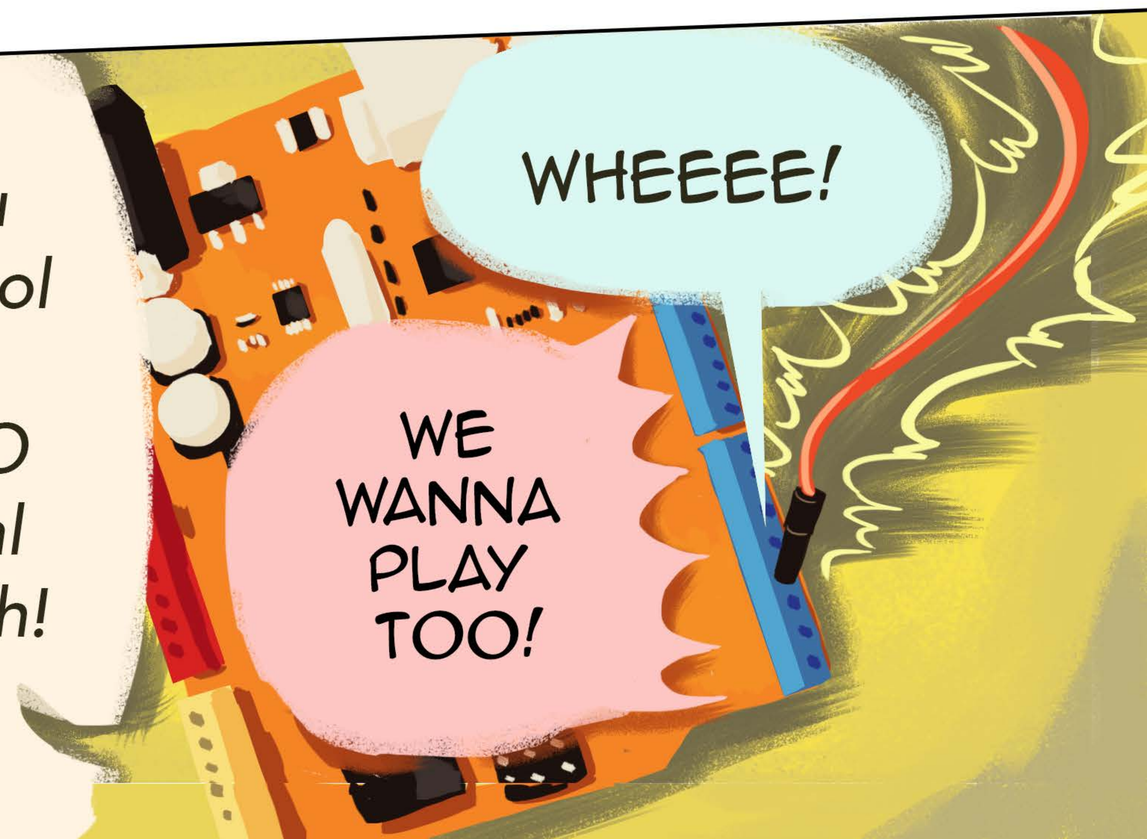
The digitalWrite (3, LOW); sets pin 3 to be 0 volts, turning the LED off. Notice the delay(1000) command telling the UNO to wait again. Without it, the void loop() will restart immediately, and the LED will turn on before you even notice it was off.



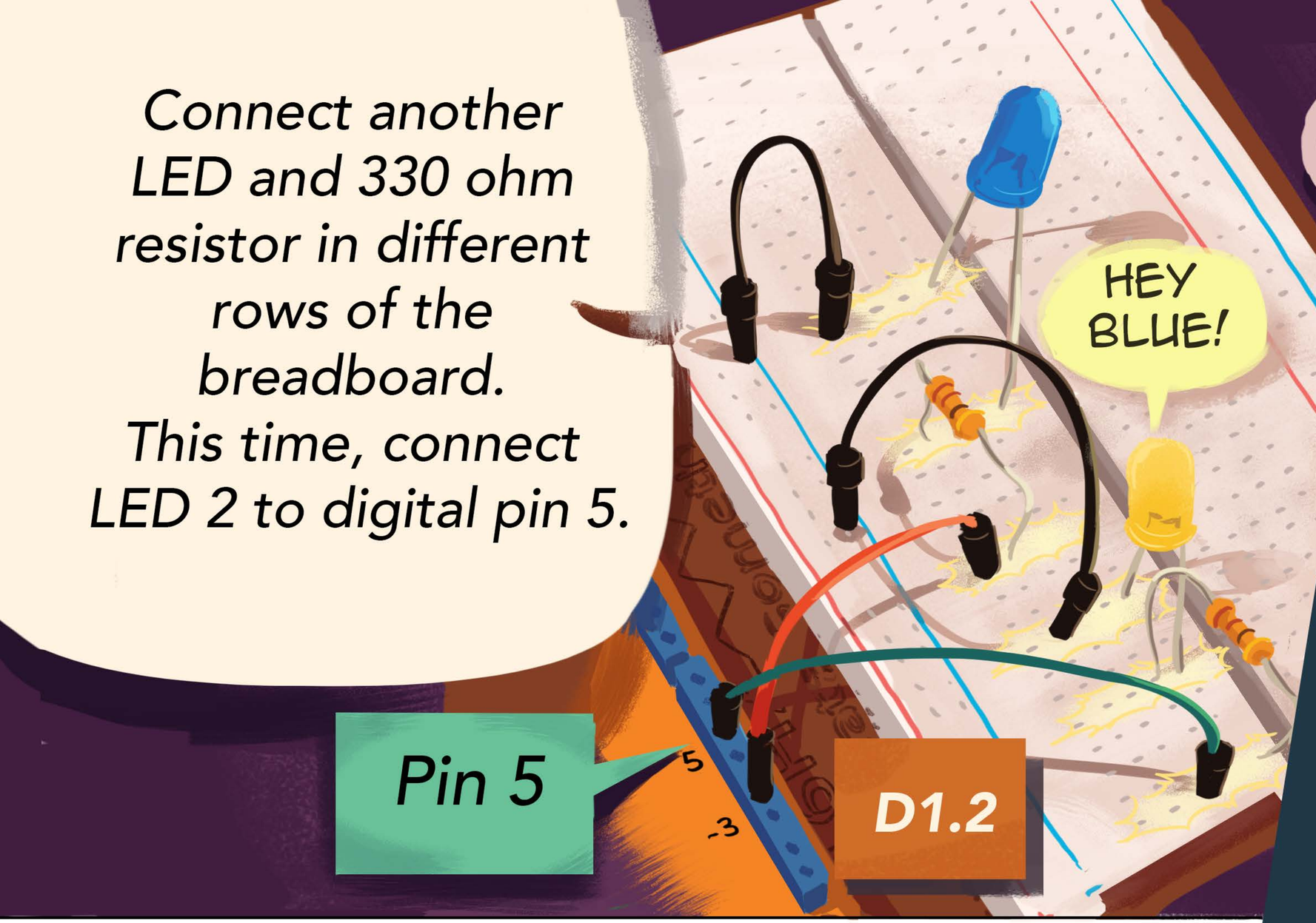
The UNO runs through lines of code hundreds of thousands of times per second! Use delay() to stop it at key points in your code.

You can now control an LED with your microcontroller!

But what if you wanted to control more LEDs? Luckily the UNO has more digital pins to play with!



Connect another LED and 330 ohm resistor in different rows of the breadboard. This time, connect LED 2 to digital pin 5.



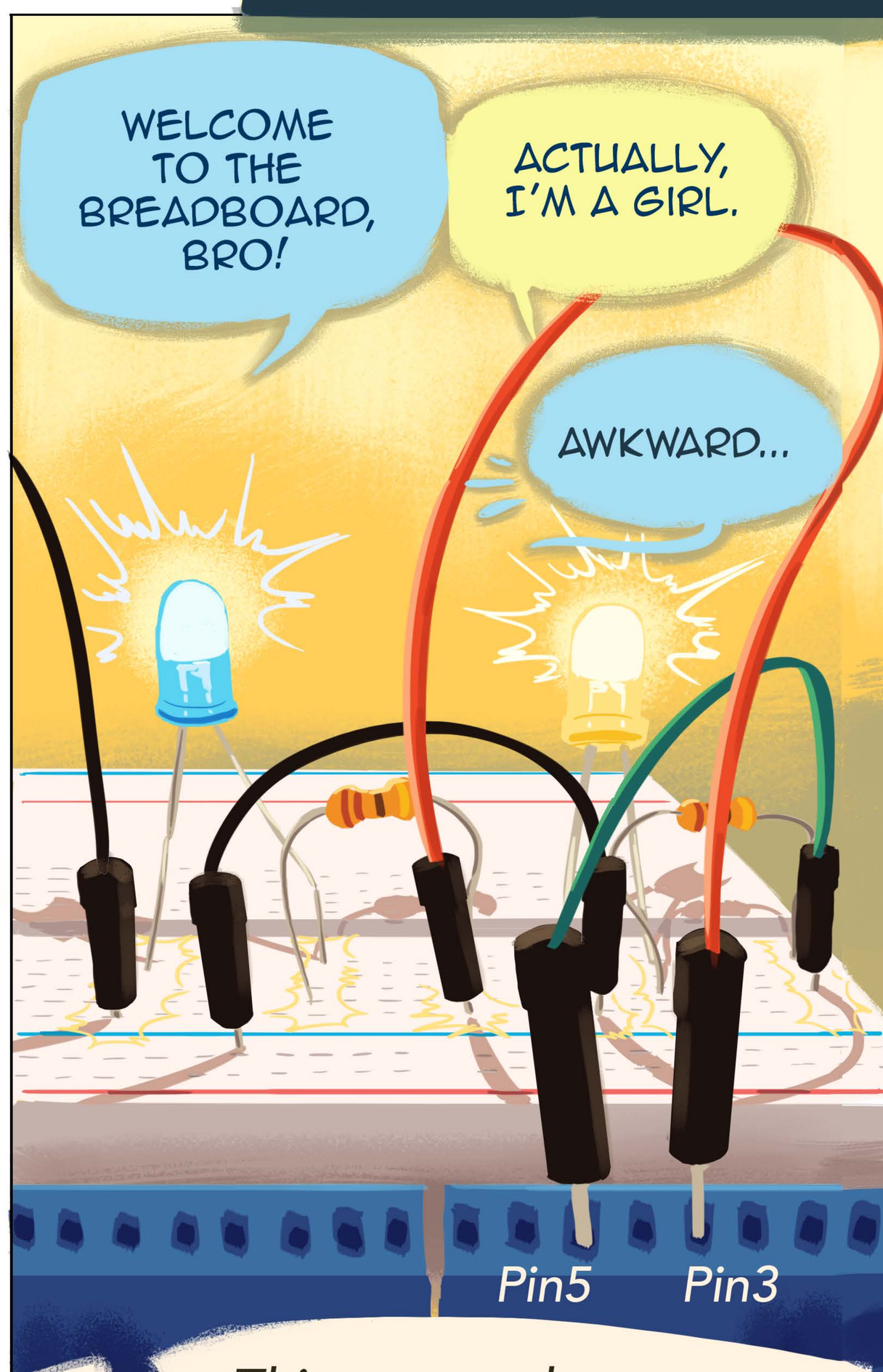
Pro-Tip:

It's a good idea to disconnect your UNO from your computer when tweaking your circuits. It protects your UNO and your computer from accidental short circuits and crossed wires.

Now we just need to alter our code to control both LEDs at once. Upload this code and watch what happens.

C1.2

```
void setup(){
      pinMode(3, OUTPUT);
      pinMode(5, OUTPUT);
}

void loop(){
      digitalWrite(3, HIGH);
      digitalWrite(5, LOW);
      delay(1000);
      digitalWrite(3, LOW);
      digitalWrite(5, HIGH);
      delay(1000);
}
```

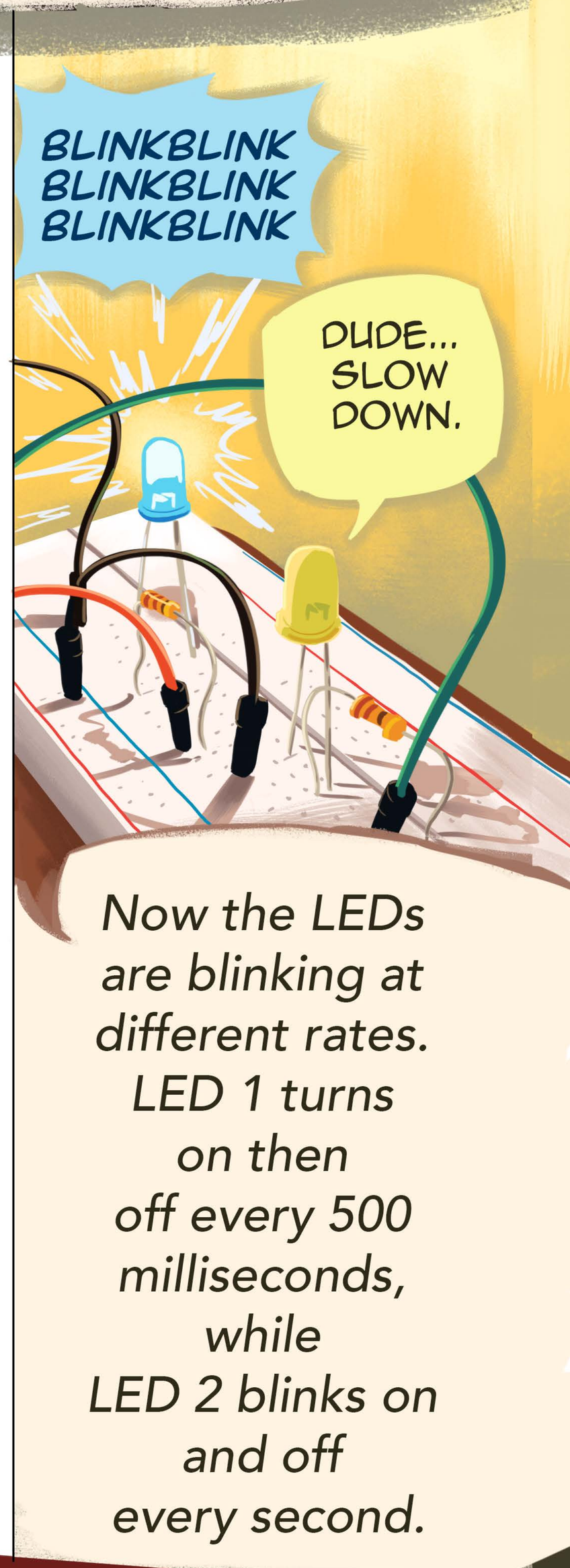


This new code blinks both LEDs in an alternating pattern. The UNO is capable of controlling each pin independently!

The code below uses a different pattern of delays. Upload it and see what it does.

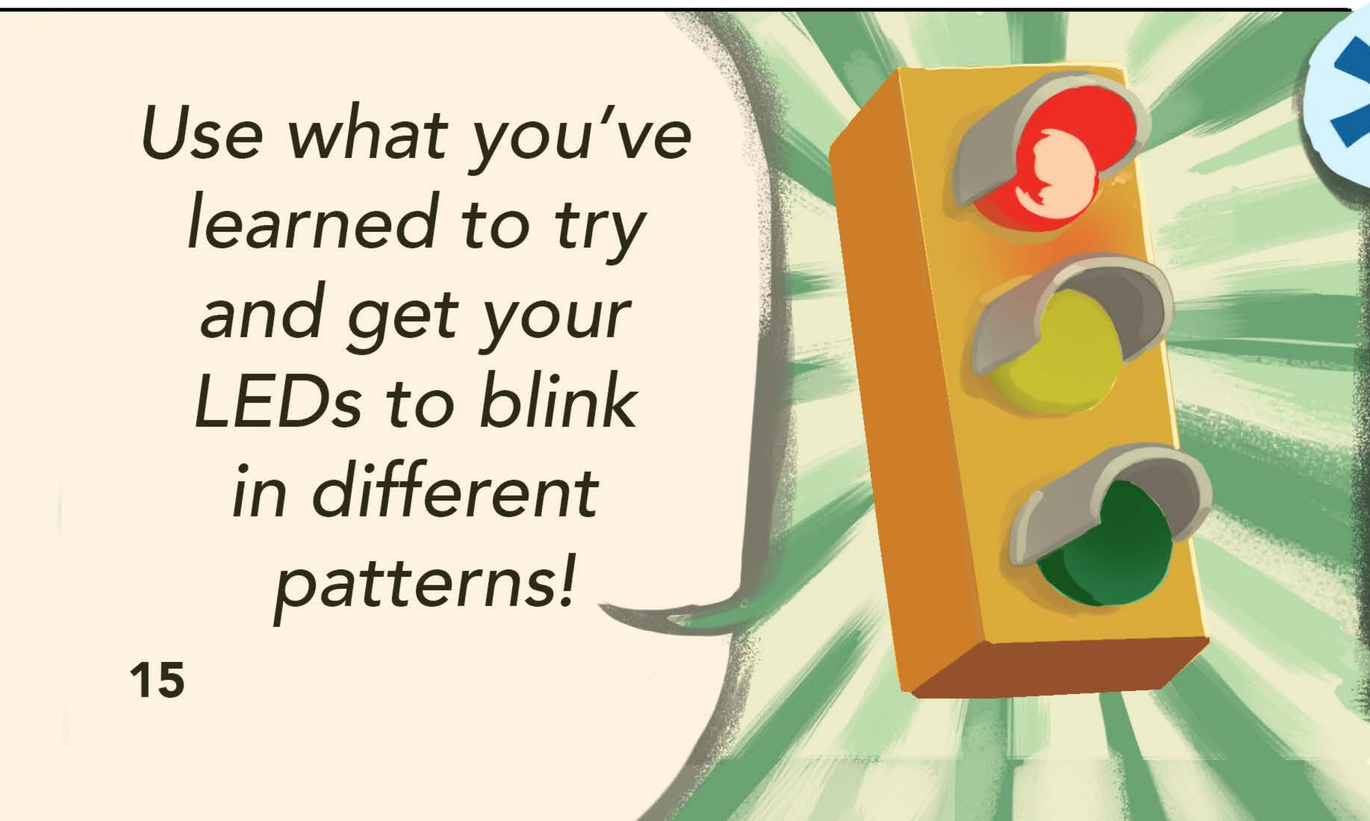
C1.3

```
void setup(){
    pinMode(3, OUTPUT);
    pinMode(5, OUTPUT);
}

void loop(){
    //LED 1: ON, LED 2: ON, then wait
    digitalWrite(3, HIGH);
    digitalWrite(5, HIGH);
    delay(250);

    //LED 1: OFF, then wait
    digitalWrite(3, LOW);
    delay(250);

    //LED 1: ON, LED 2: OFF, then wait
    digitalWrite(3, HIGH);
    digitalWrite(5, LOW);
    delay(250);

    //LED 1: OFF, then wait
    digitalWrite(3, LOW);
    delay(250);
}
```



Helpful Hint:

Adding a "//" before some code makes it a "comment". Comments are ignored by the UNO and are nice ways to leave notes for yourself.



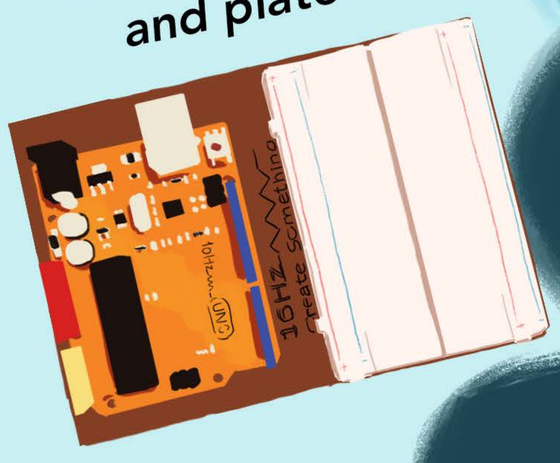
Challenge:

1a. *Can you make an LED blink twice before a long pause?*

1b. *Can you recreate a traffic light pattern with red, yellow, and green LEDs?*

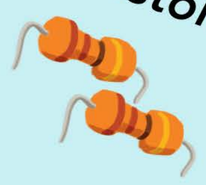
TUTORIAL 2:
DIGITAL READ

16Hertz UNO, breadboard, and plate



10K Ohm Resistor

330 Ohm Resistors



LET'S LEARN ABOUT...

Using the `digitalRead()` command to read the state of a button

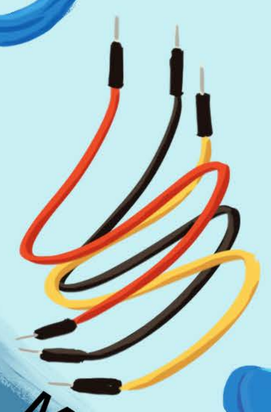
MATERIALS:

- ☐ 16Hertz UNO + Breadboard
- ☐ USB cable
- ☐ Male-Male Jumper wires
- ☐ Two LEDs
- ☐ 10 Kilo-ohm resistor
- ☐ Two 330 Ohm Resistors
- ☐ Computer & Arduino IDE
- ☐ Push Button

USB

Arduino IDE

Male- Male Jumper Wires

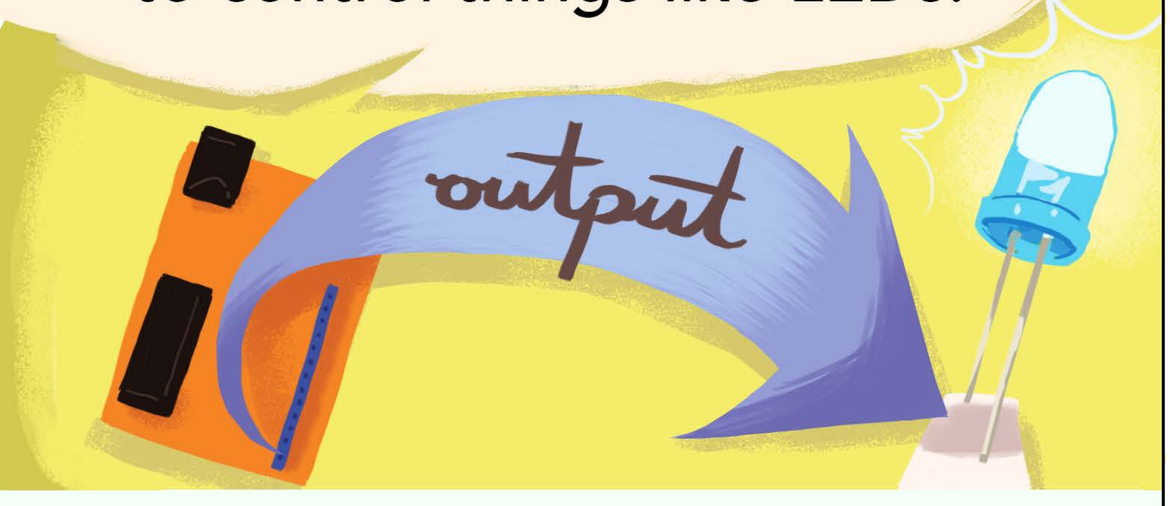


LEDs

Push Button

Let's make the UNO do things when you press a button!

The UNO's digital pins can be used as OUTPUTS to control things like LEDs.

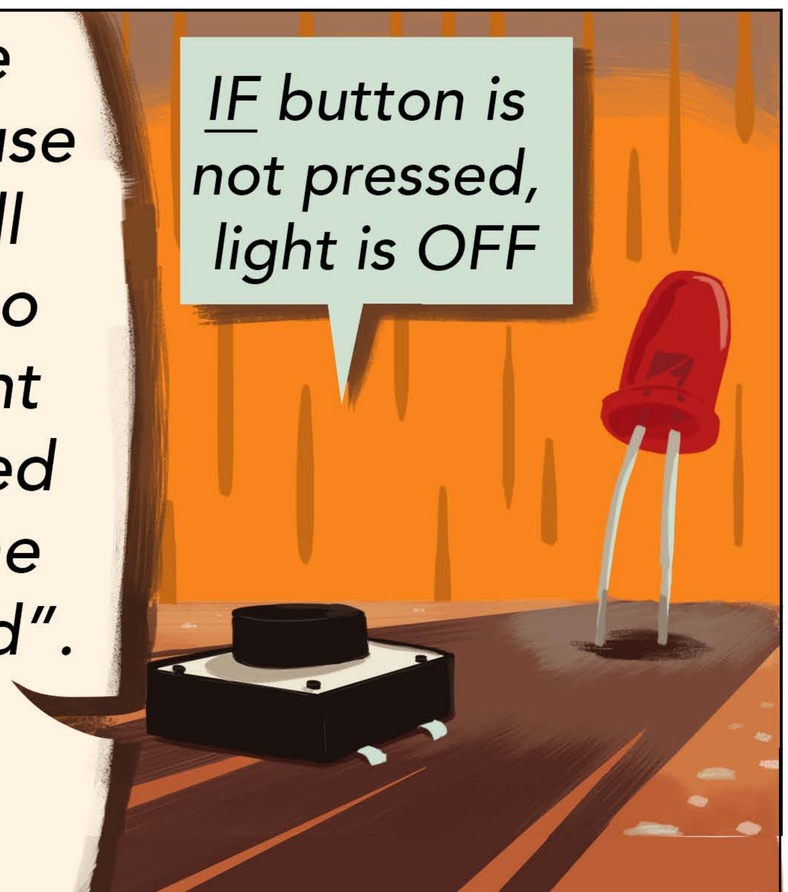


`digitalWrite()` ⟷ OUTPUT

However, the pins can also be INPUTS, to read things like buttons and sensors. For INPUTS we use the `digitalRead()` command.

`digitalRead()` ⟷ INPUT

Inputs are great because we can tell the UNO to do different things based on what the inputs "read".



Our next goal will be to make an LED turn on if a button is pressed and be off if it's not pressed.

C2.1

Let's try reading what a pin receives when using `digitalRead()`. Upload this new code to your UNO.

```
void setup(){
  Serial.begin(9600);
  pinMode(7, INPUT);
  Serial.println("Hello!");
}
void loop(){
  Serial.println( digitalRead(7) );
  delay(50);
}
```

This starts communication be--tween the UNO and the com--puter so we can send messages back to our screen. The 9600 is how fast the UNO is talking.

This sets digital pin 7 to be an INPUT.

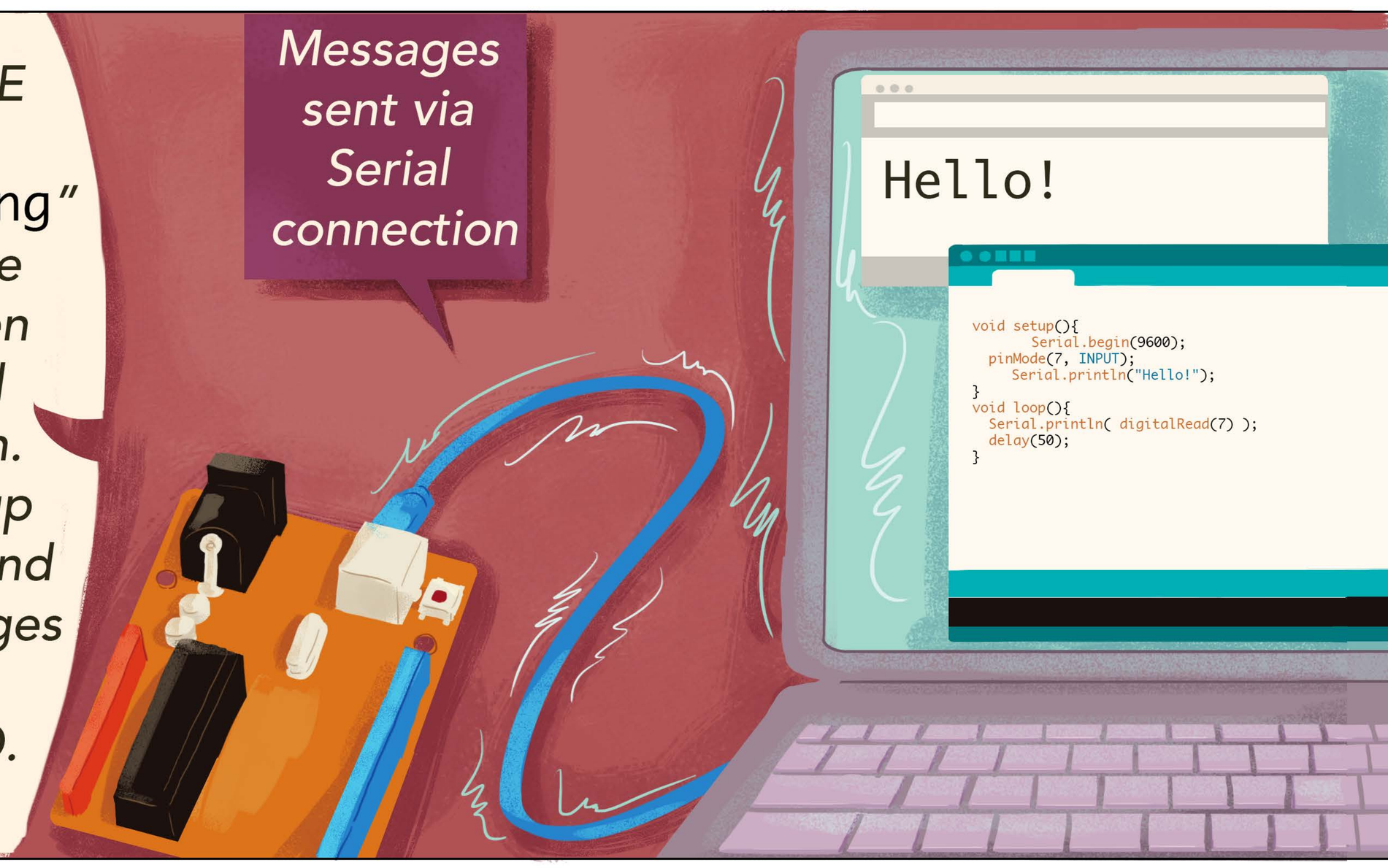
This sends a message of whatever is between the first set of parentheses back to the computer so we can read it. The quotes mean the message will be sent as text characters.

This slows down the sketch a tiny bit to make it easier to read the results.

This part returns what the input pin reads as a number 0 or 1.

After uploading, we can see the messages the UNO sends by clicking the Serial Monitor button.

Wait for the IDE to show the "`Done Uploading`" message on the bottom left then click the Serial Monitor button. This will open up a new window and show the messages being sent from the UNO.



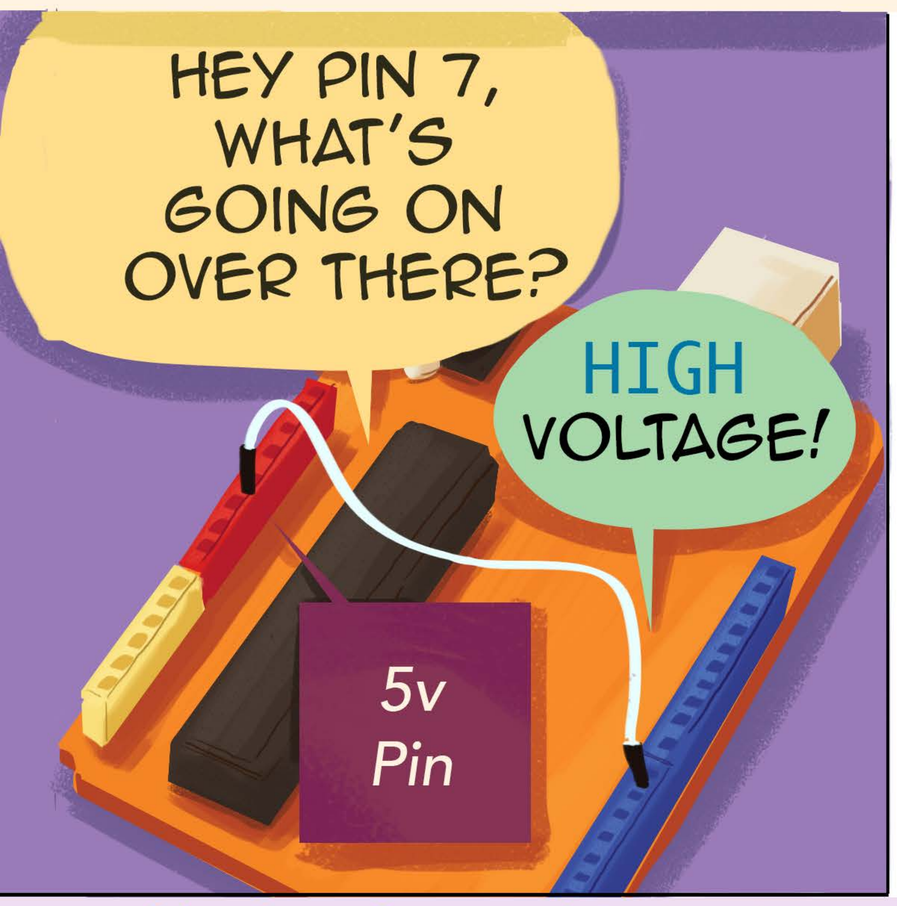
Plug one end of a jumper wire into digital pin 7 and the other end into the 5V pin on the other side of the board.

What do you see in the Serial Monitor? What happens if you remove the wire from the 5V pin and plug it into the GND pin?

You may have noticed the messages in the Serial Monitor bounce between 0 and 1 erratically when the other end of the wire wasn't connected to anything. Let's see why this happens...

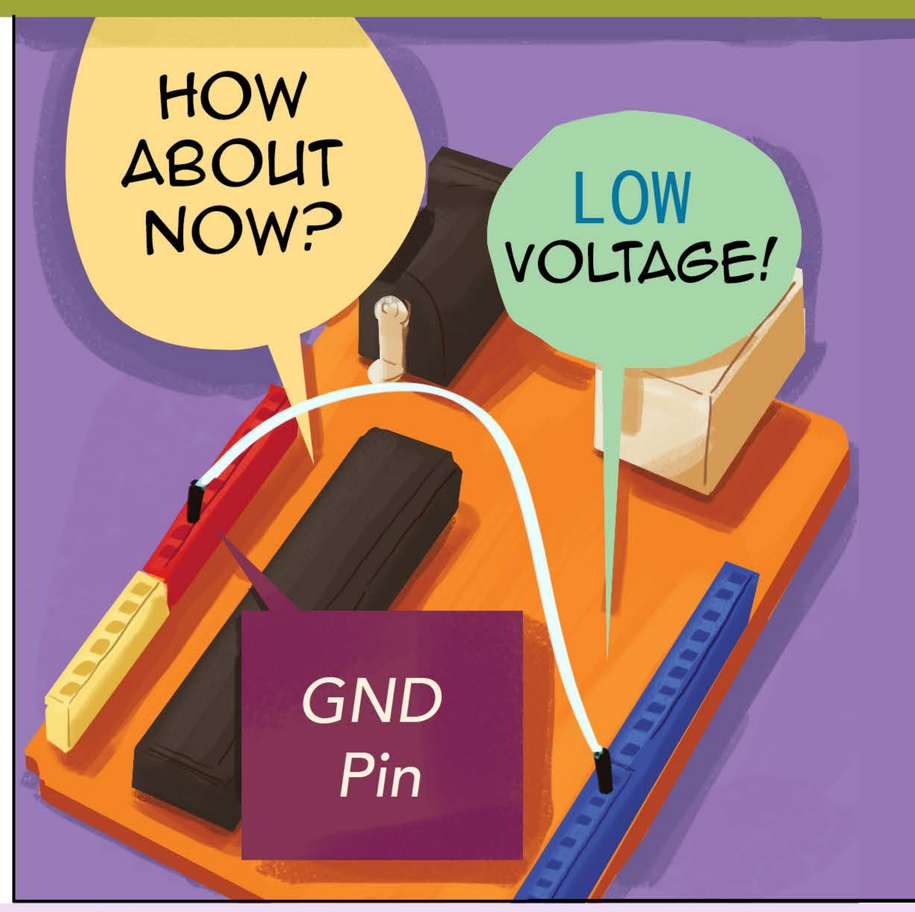
The `digitalRead()` *command tells the UNO to read what is happening at the pin.*

Digital `INPUT` *is all about* `HIGH` *and* `LOW` *voltages.*



`digitalRead(7)` is `HIGH` *or* `1`

"Floating Pin"

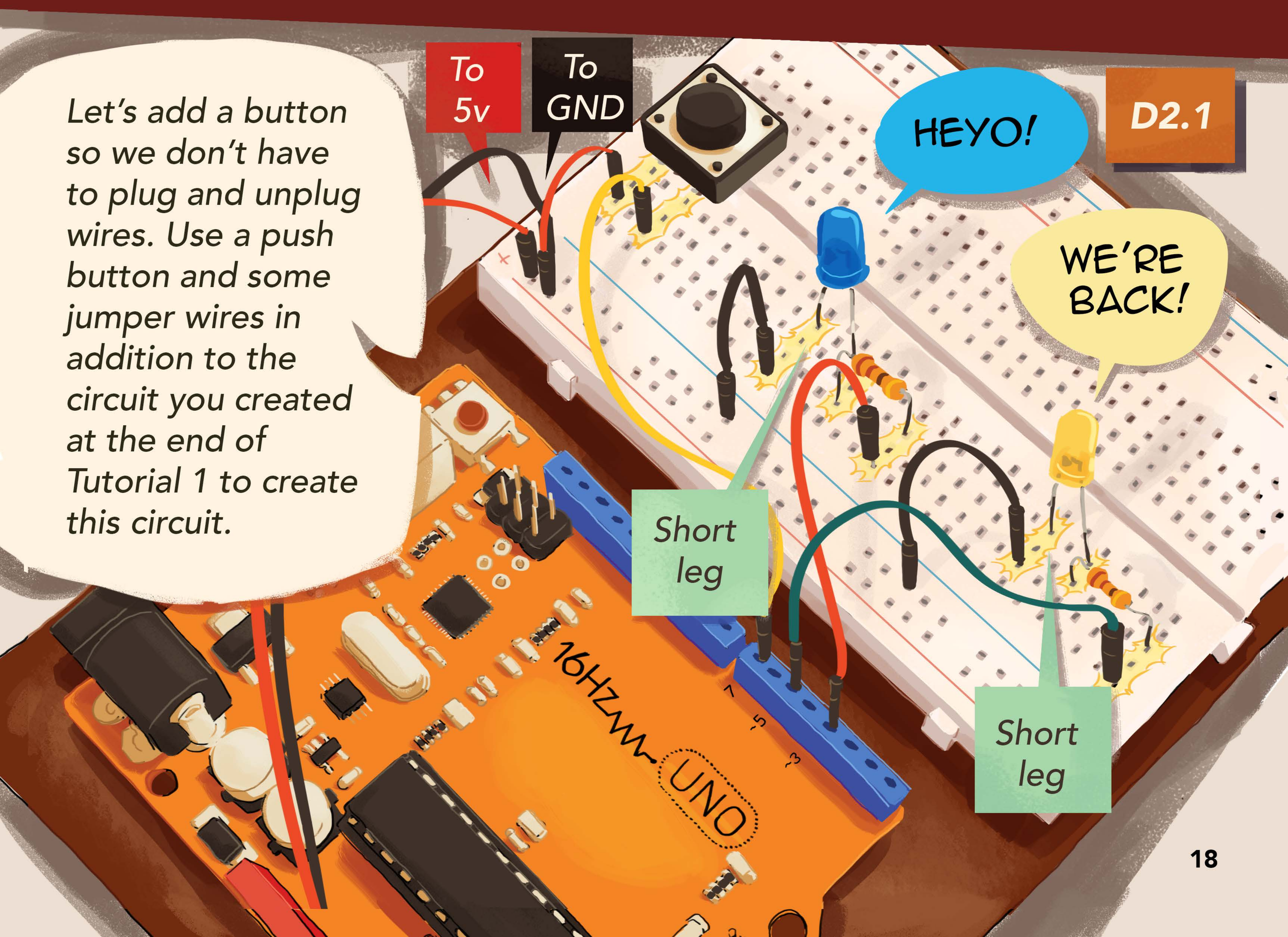


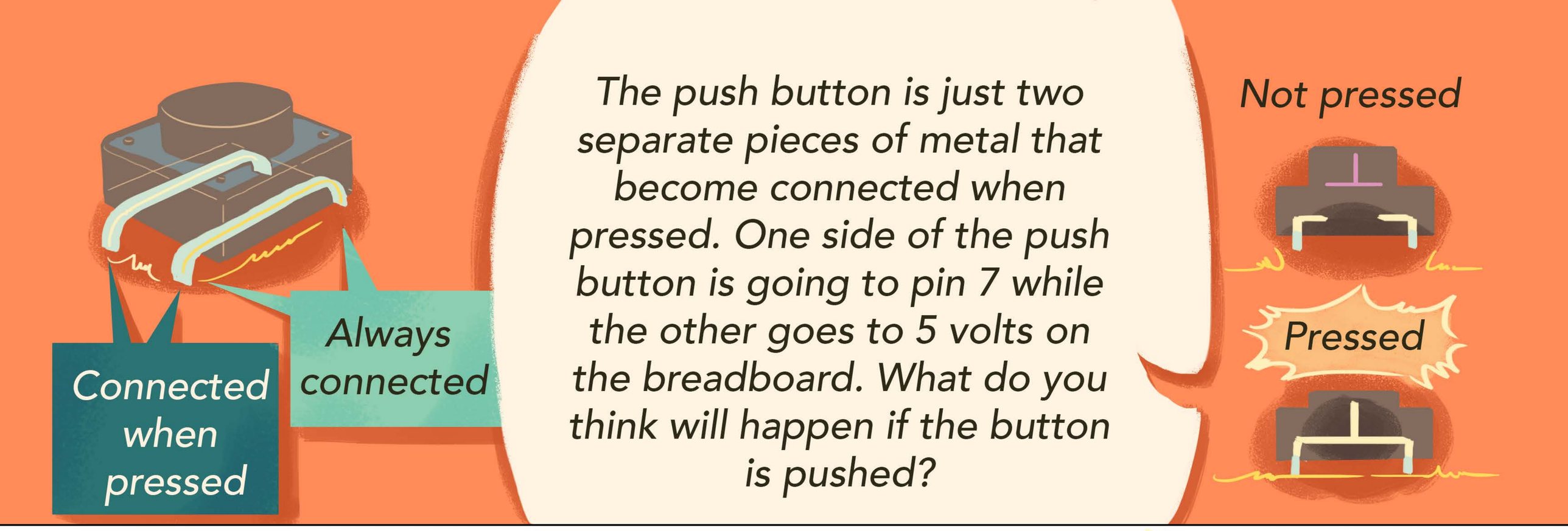
`digitalRead(7)` is `LOW` *or* `0`

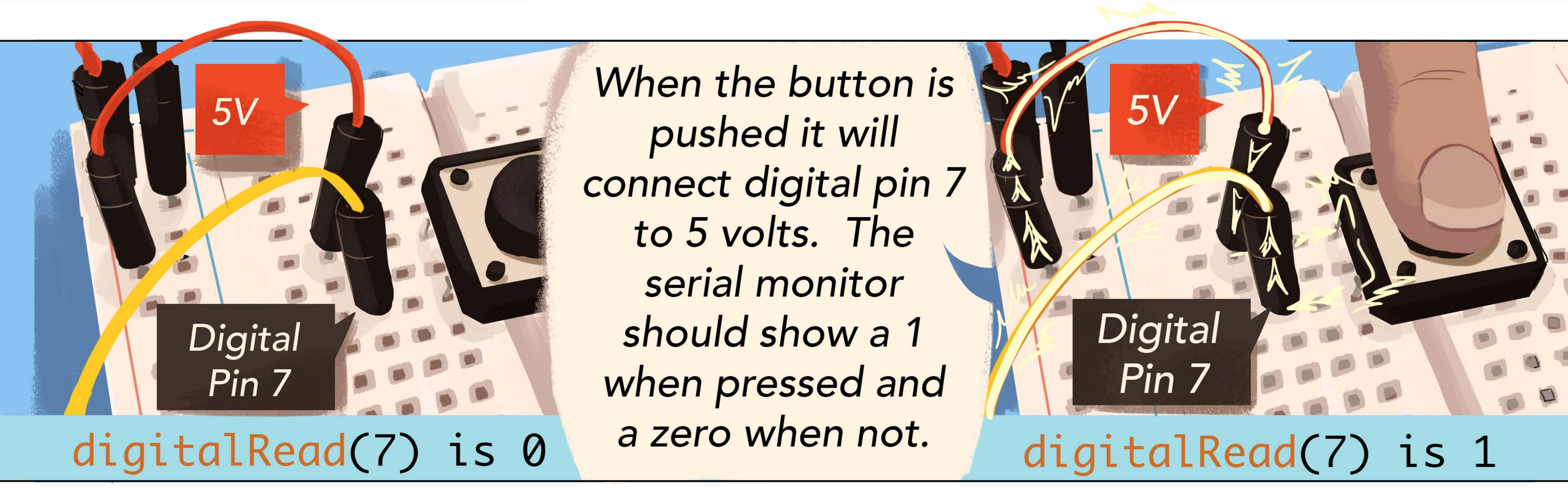
Helpful Hint:

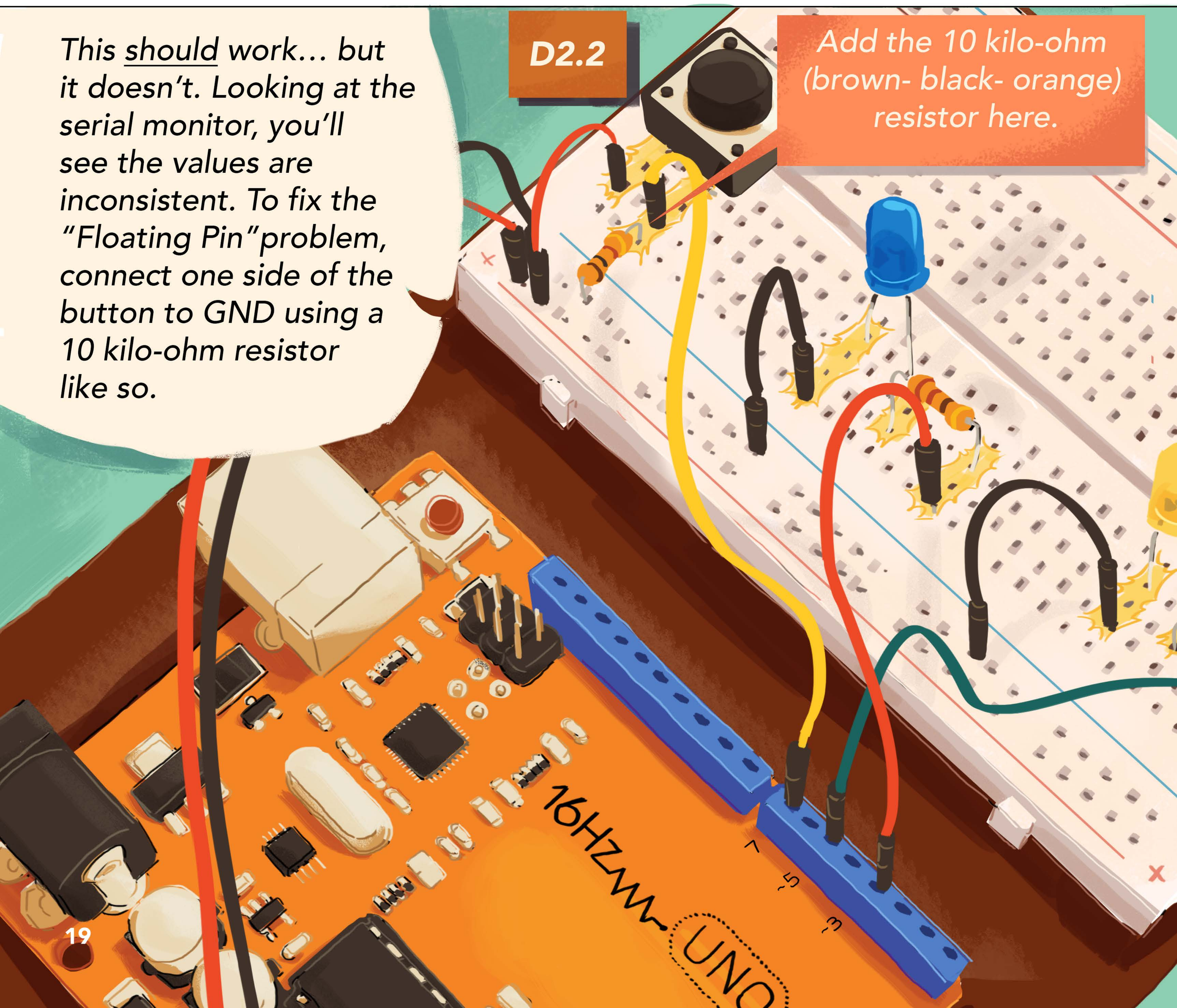
Input pins are very sensitive to small currents. This means even random electrical noise will make floating pins give weird readings. We'll learn to fix this later.

Let's add a button so we don't have to plug and unplug wires. Use a push button and some jumper wires in addition to the circuit you created at the end of Tutorial 1 to create this circuit.

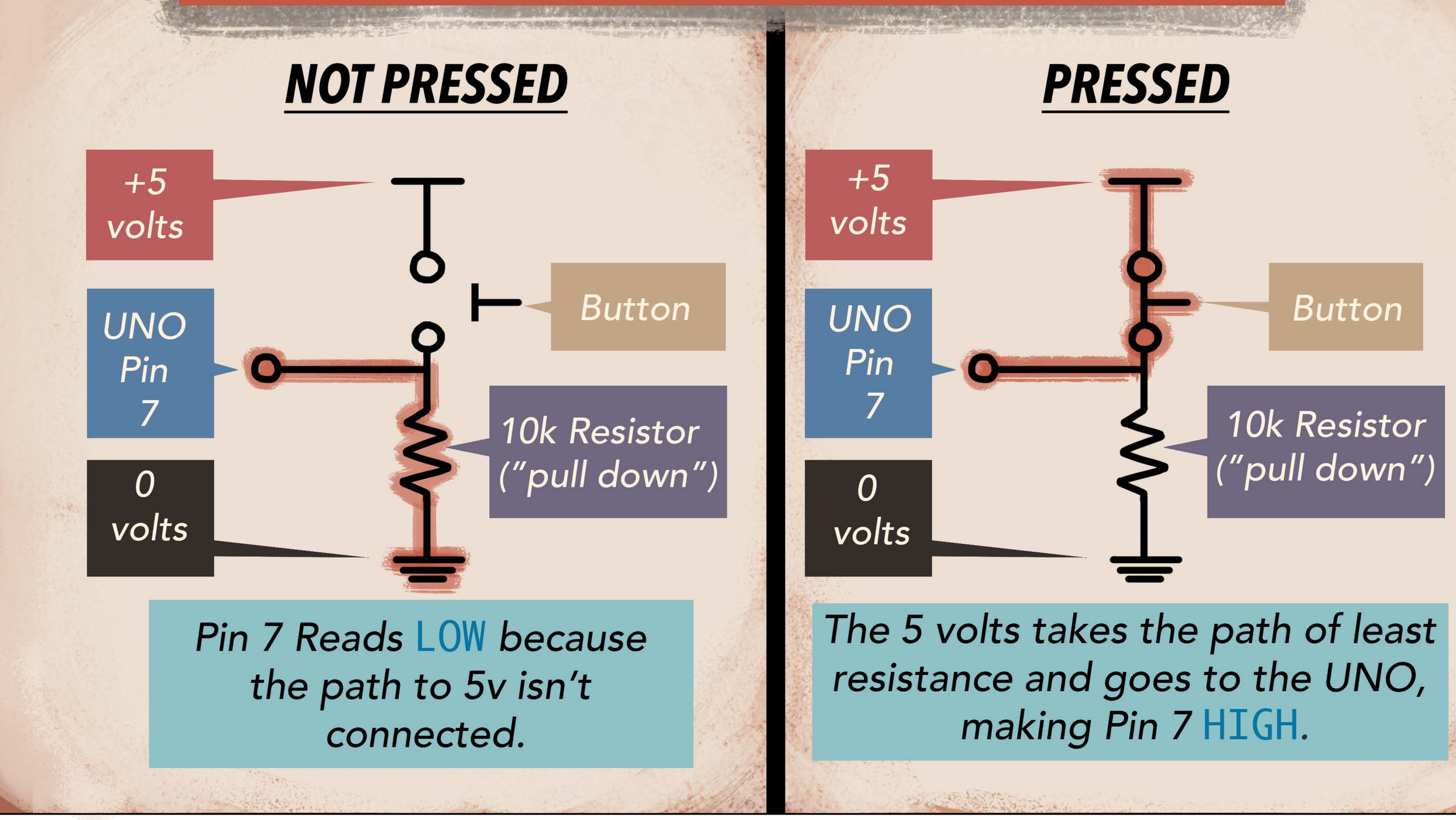



Connected when pressed
Always connected
The push button is just two separate pieces of metal that become connected when pressed. One side of the push button is going to pin 7 while the other goes to 5 volts on the breadboard. What do you think will happen if the button is pushed?
Not pressed
Pressed


5V
Digital Pin 7
digitalRead(7) is 0
When the button is pushed it will connect digital pin 7 to 5 volts. The serial monitor should show a 1 when pressed and a zero when not.
5V
Digital Pin 7
digitalRead(7) is 1


This should work… but it doesn't. Looking at the serial monitor, you'll see the values are inconsistent. To fix the "Floating Pin"problem, connect one side of the button to GND using a 10 kilo-ohm resistor like so.
D2.2
Add the 10 kilo-ohm (brown- black- orange) resistor here.
16HZ
UNO
7
~5
~3

A resistor used in this fashion is known as a "pull down" resistor and it ensures the pin is always either 5V or 0V and never floating. Check out this schematic view:



The hardware is now set, so let's add some code that will allow the UNO to make decisions based on the button's state. For this we will use an "if/else statement". `if()` statements let you run different sections of code IF a specific condition is true or false.

This statement is the condition the UNO will check to make a decision.

```
if(digitalRead(7) == 1 ){
    //do code between these curly braces
    //if the button is pressed
} else {
    //do code between these curly braces
    //if the button is NOT pressed
}
```

Helpful Hint:

In the Arduino language, "=" and "==" mean different things. The single "=" is TELLING something to be equal to something else. The double equal sign is ASKING a question like "Is `digitalRead(7)` equal to 1?" The `if()` statement will check to see if this answer is true or false then run the right code for it. That's why the "=" won't work.

Upload the following code to your UNO. Note how it uses a combination of the new `digitalRead()` *and if/else functions.*

C2.2

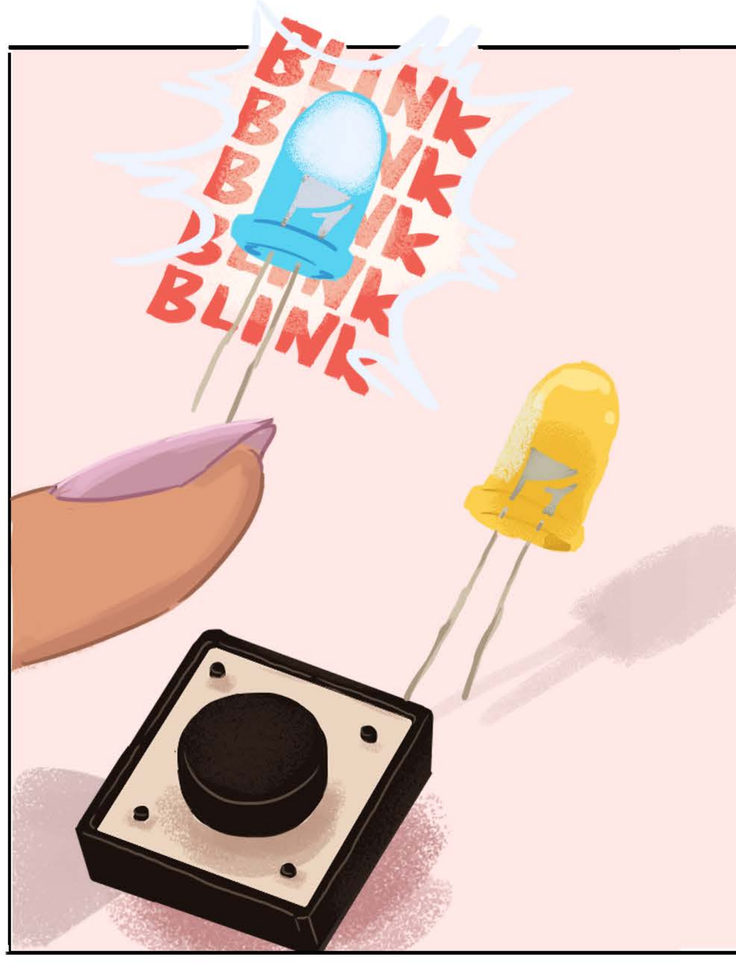
```
void setup(){
    pinMode(3, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(7, INPUT);
}

void loop(){
    //Is our button pressed?
    if(digitalRead(7) == 1){
        //Turn one LED on and the other off
        digitalWrite(3, HIGH);
        digitalWrite(5, LOW);
    }else{
        //Switch the states of the LEDs
        digitalWrite(3, LOW);
        digitalWrite(5, HIGH);
    }
}
```

Press the button and see what happens! You'll notice the UNO is running two different bits of code depending on if the button is pressed or not pressed.

Challenge:

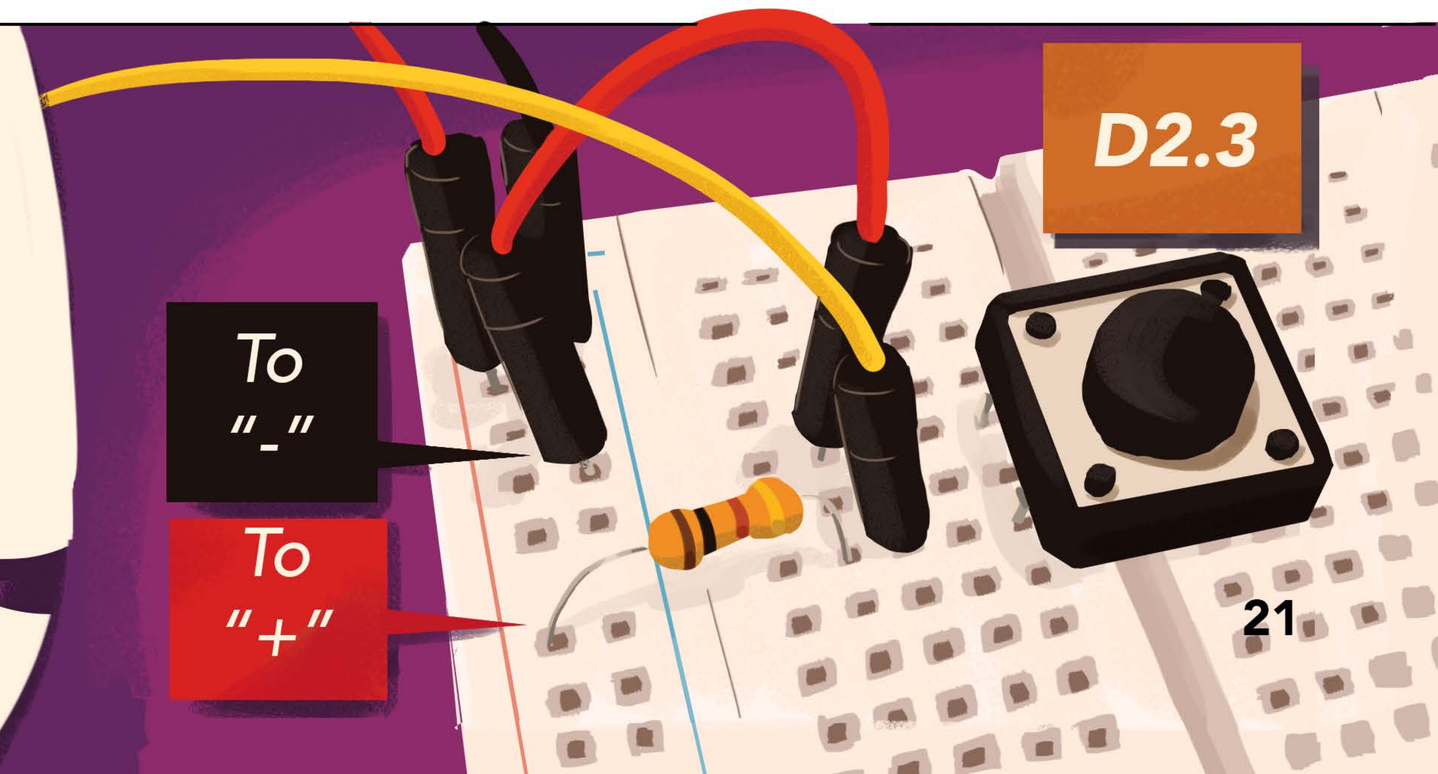
2a. *When the button is pressed, have the UNO blink one LED rapidly while the other is off. When released, reverse the behavior.*

Hint: *Use short delays to make sure your UNO reads the button often enough.*

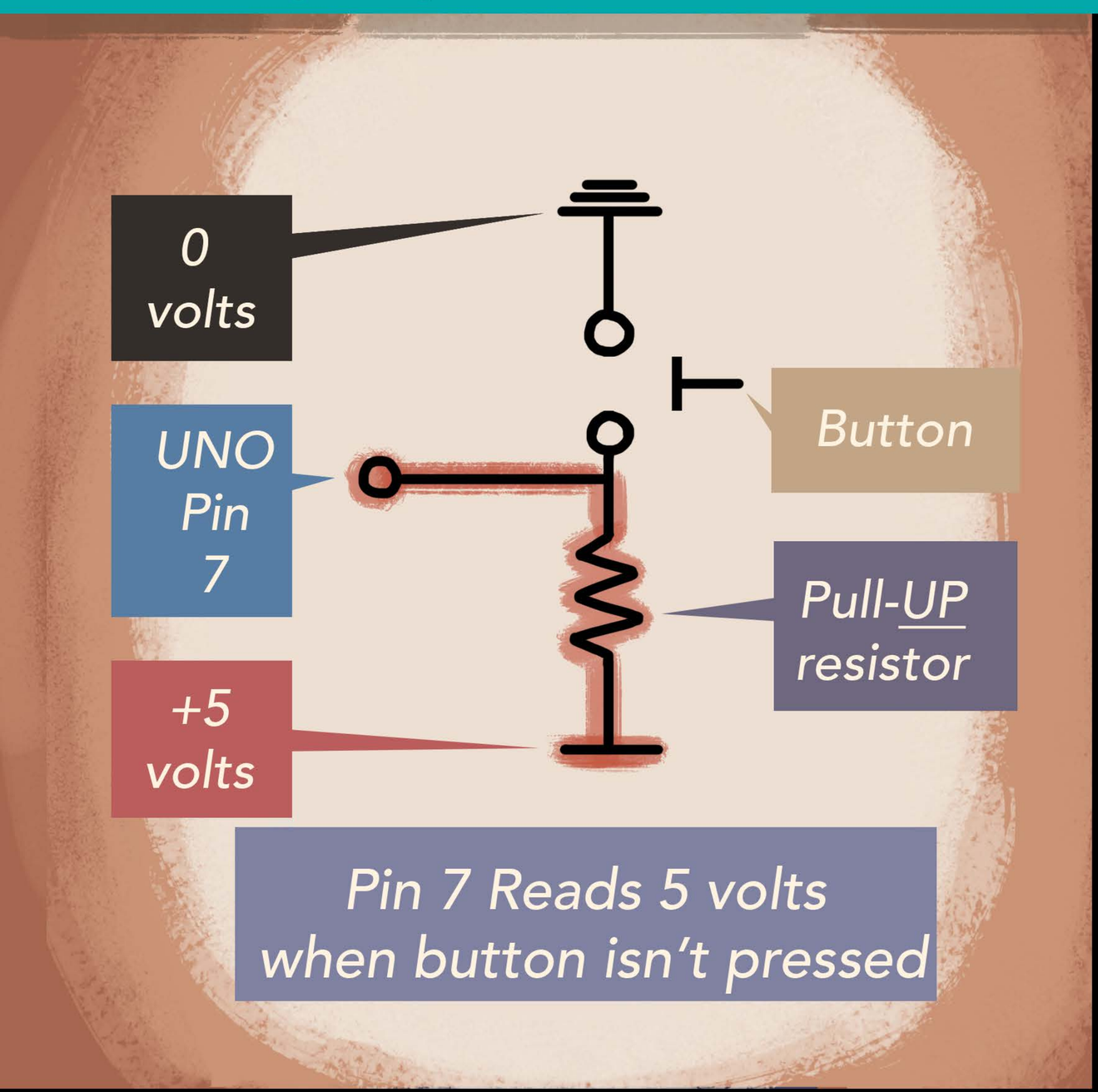


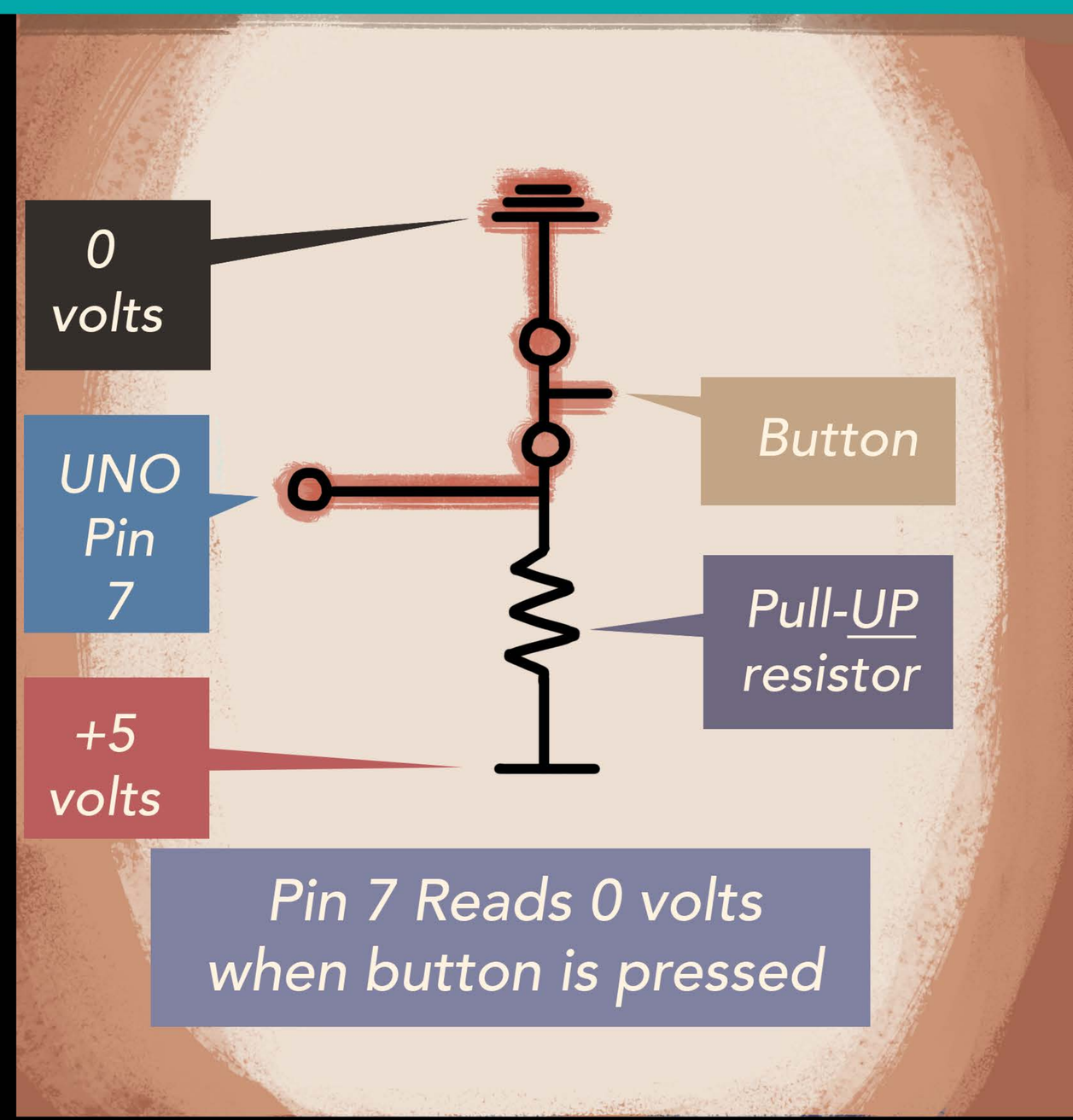
Lets try something crazy! Move the jumper wire that connects the button to the 5V rail to the ground rail. Also, move the 10K resistor from the ground rail to the 5V rail just like it's shown here.

D2.3

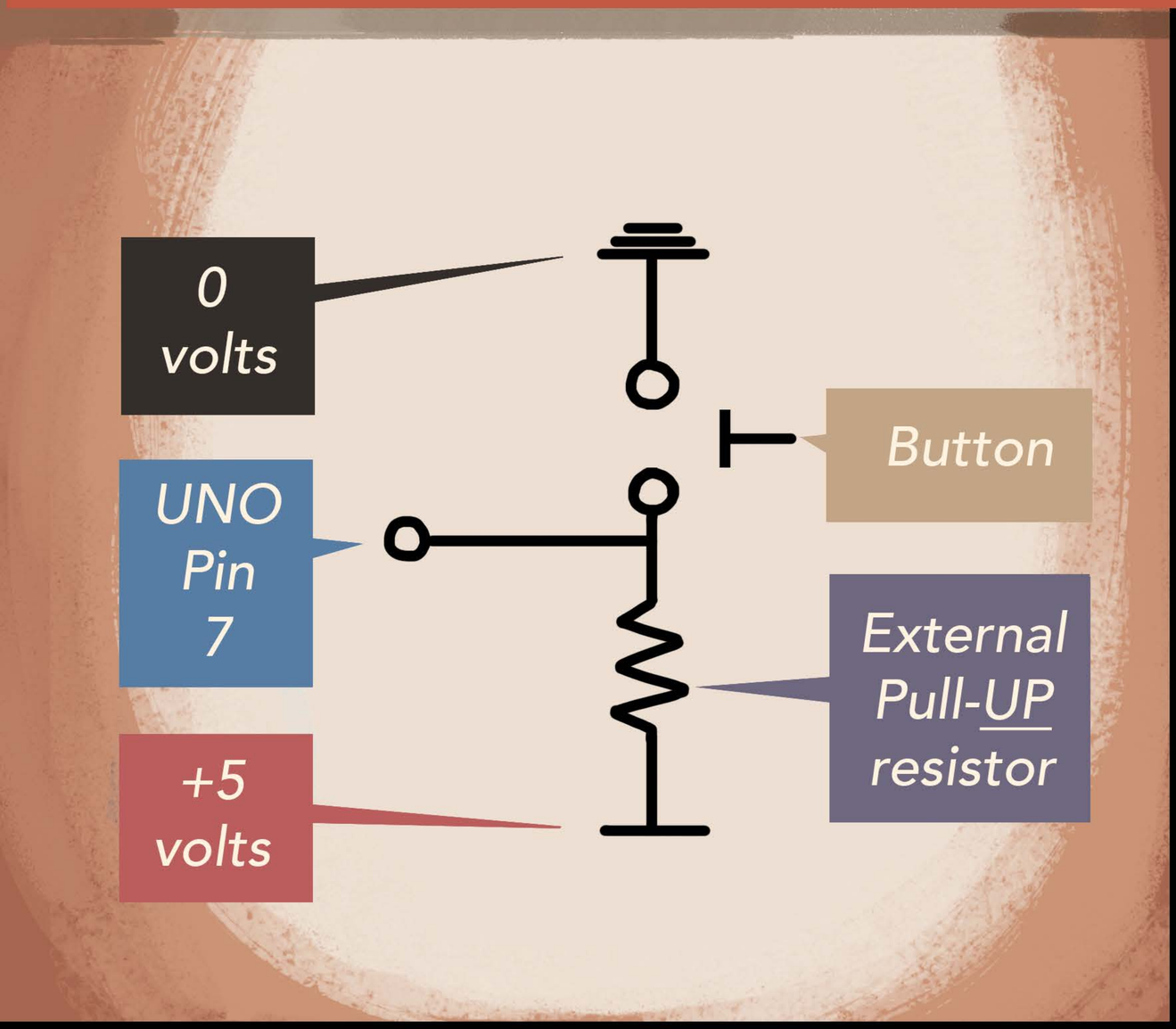


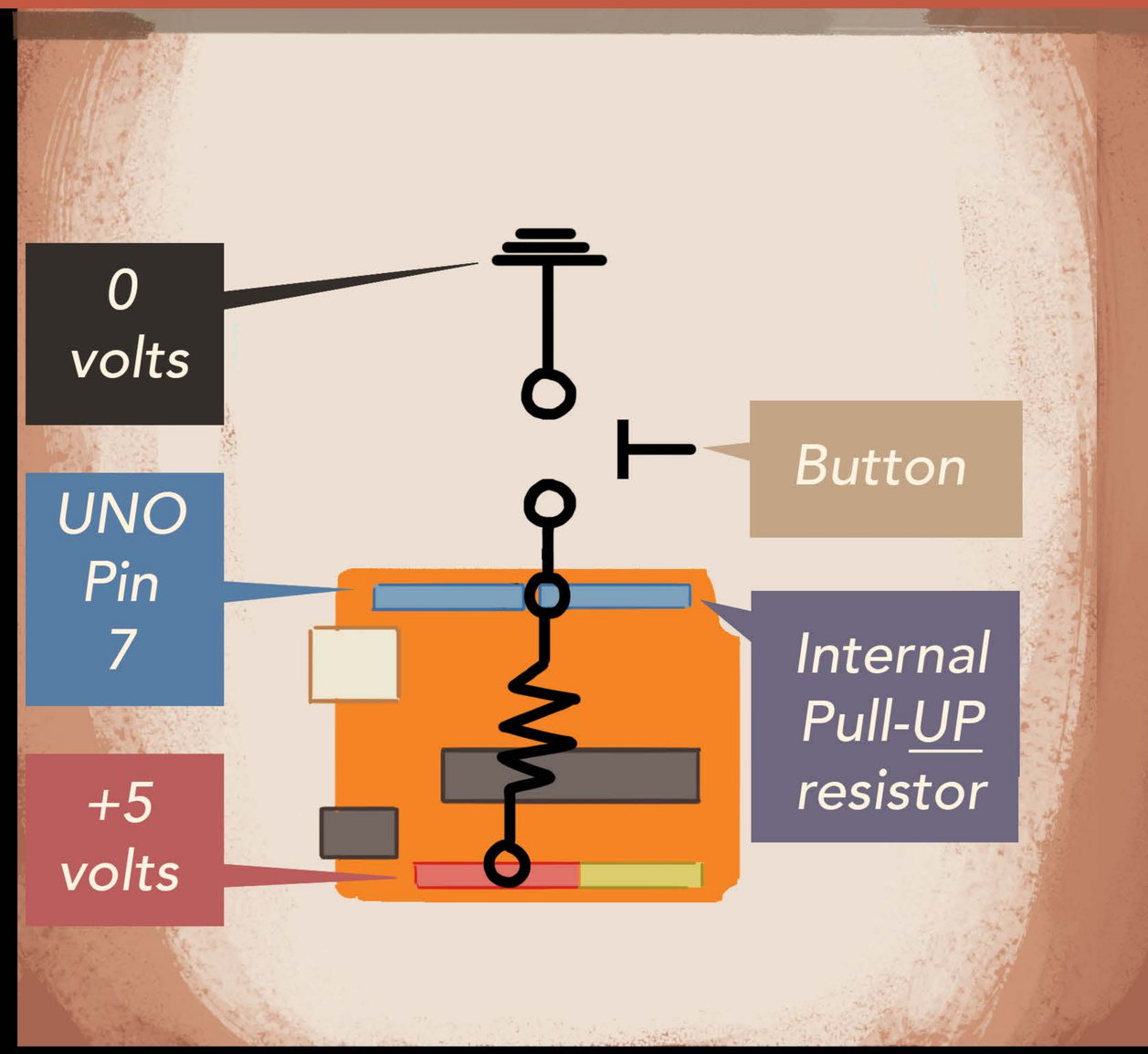
Notice how the LEDs have reversed their behavior! This is because it's pulling the pin up to 5 volts until we press the button to connect it directly to ground.



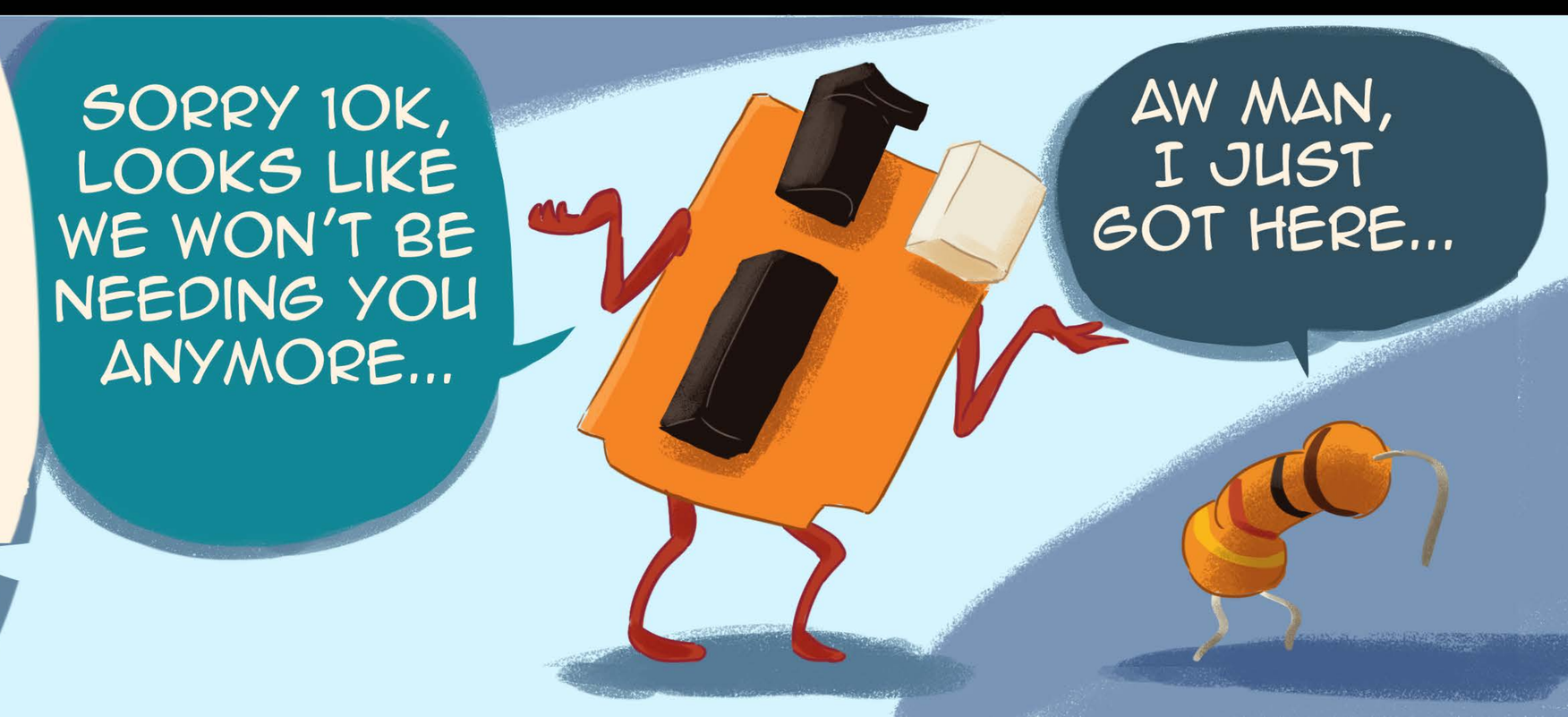


As it turns out, your UNO already has convenient pull-up resistors for all your digital pins embedded inside it!

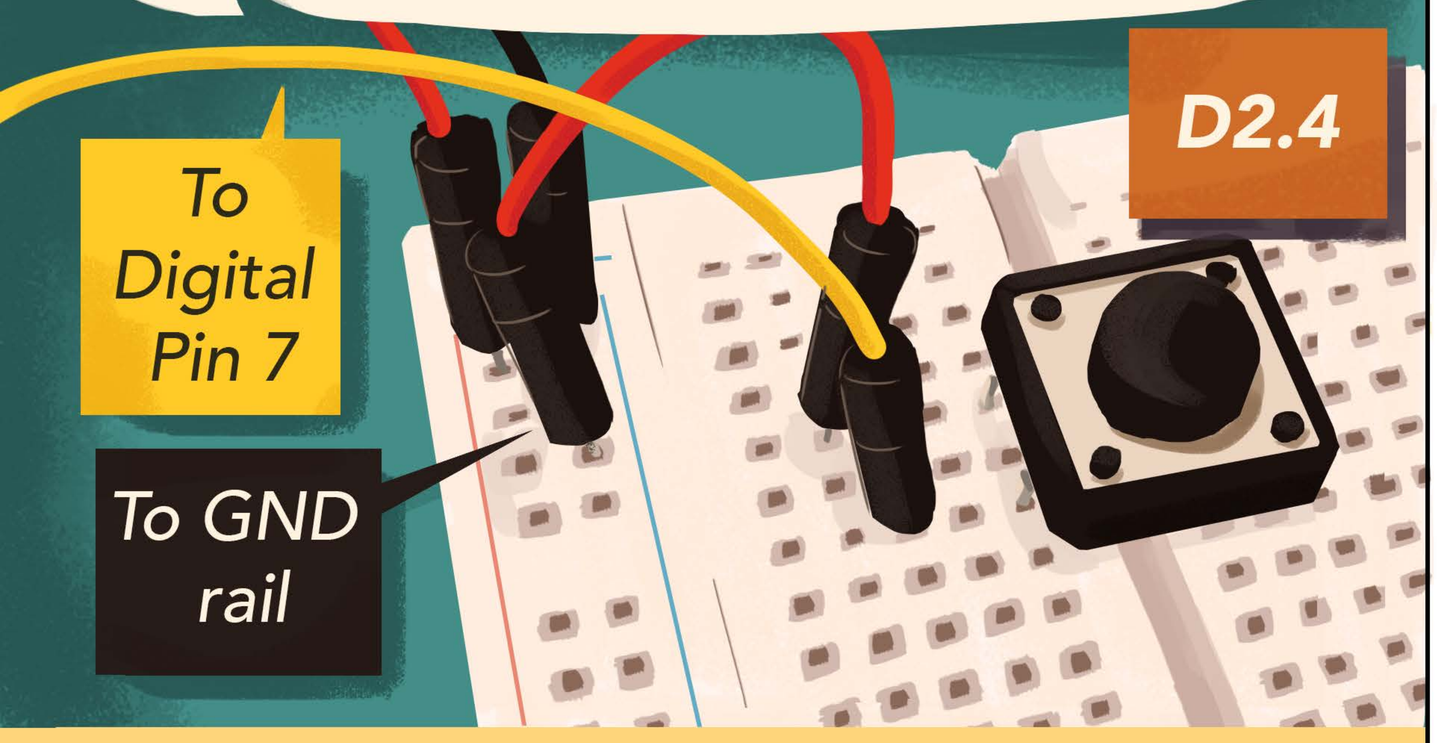




Since the UNO has its own, internal pull-ups, lets activate them so we don't need to use our own. This will make the wiring simpler in the end.



Remove the 10K resistor from your breadboard and put it aside. Connect one side of the button to the ground rail and the other side to pin 7 like so:



Go into your code and replace the line that says "`pinMode(7, INPUT);`" and replace it with "`pinMode(7, INPUT_PULLUP);`"

And that's all there is! The code is almost the same but the wiring is much simpler. This will make it easier if we want to add multiple buttons.

However, there's a catch. The LEDs are doing the reverse of what they did when we pressed the button before. This is because pin 7 now sees 5 volts all the time and 0 volts when the button is pressed. Change this line here to fix the code.

C2.3

```
void setup(){
    pinMode(3, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(7, INPUT_PULLUP);
}

void loop(){
    //is our button pressed
    if(digitalRead(7) == 0){
      //Turn one LED on and the other off
      digitalWrite(3, HIGH);
      digitalWrite(5, LOW);
    }else{
      //Switch the states of the LEDs
      digitalWrite(3, LOW);
      digitalWrite(5, HIGH);
    }
}
```

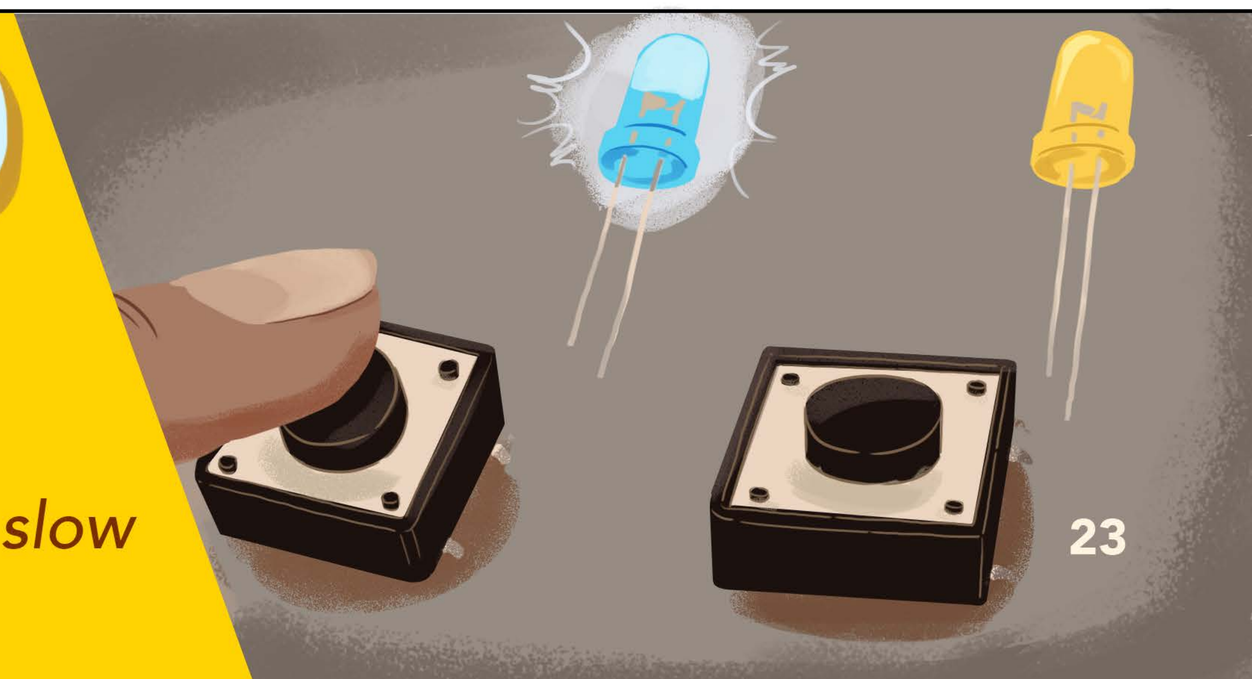
Button pressed = 0 volts = 0

Challenge:

Add a second button to your breadboard so you can...

2b. Control both LEDs individually

2c. Use one button to blink both LEDs fast and the other button to blink them slow



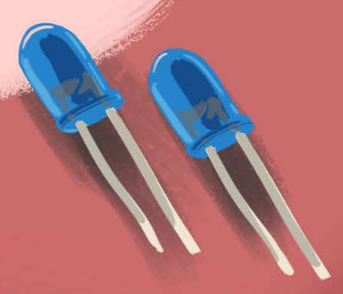
TUTORIAL 3:
ANALOG WRITE

16Hertz UNO, breadboard, and plate

USB

Male- Male Jumper Wires

2 LEDs (same color)



RGB LED

3 330 Ohm Resistors

Arduino IDE

3 Push Buttons

LET'S LEARN ABOUT...

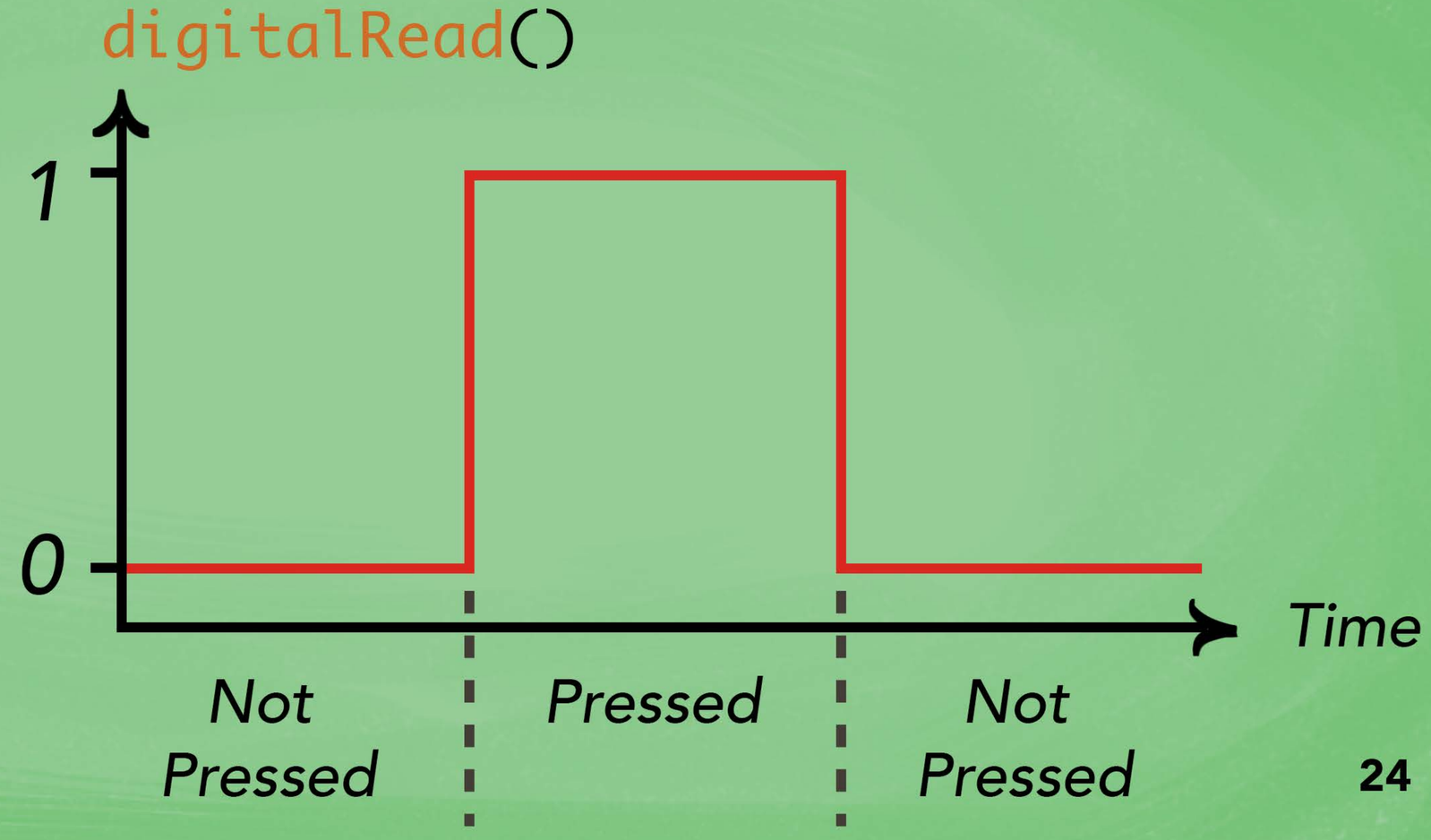
Using the `analogWrite()` *command to control an LED's brightness and make different colors*

MATERIALS:

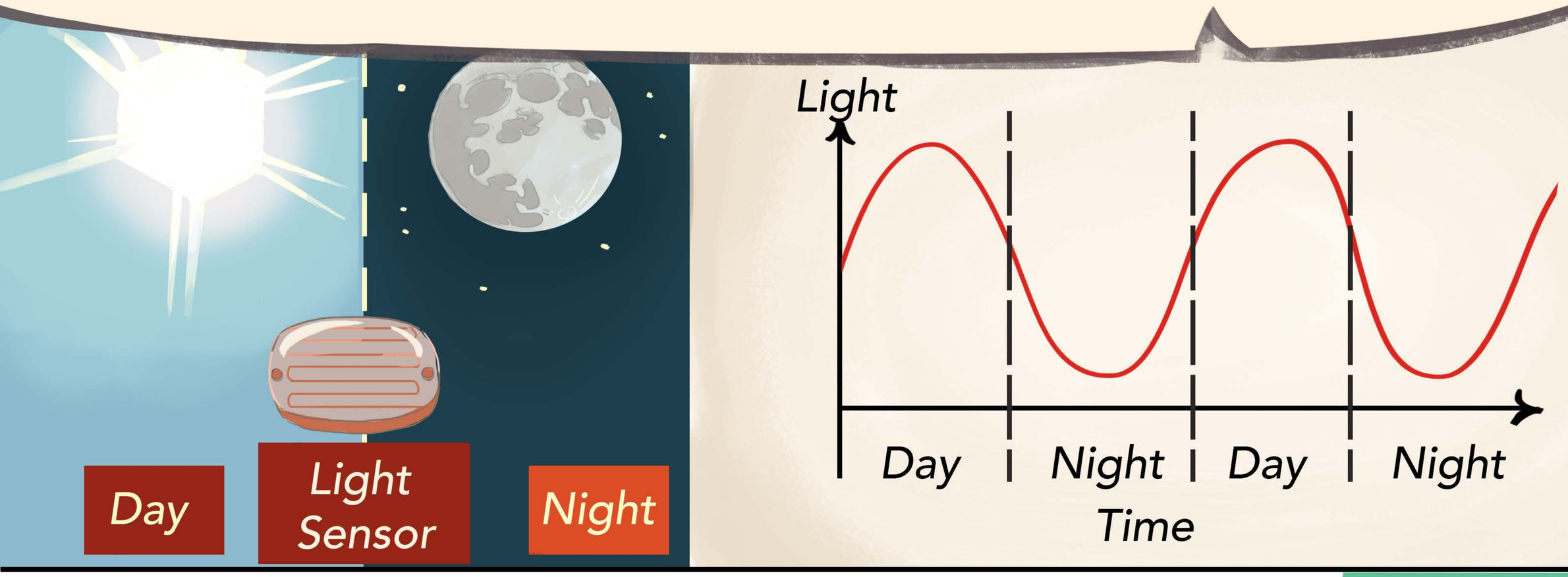
- ☐ *16Hertz UNO + Breadboard*
- ☐ *USB Cable*
- ☐ *Male/male Jumper wires*
- ☐ *2 x LEDs (same color)*
- ☐ *RGB LED*
- ☐ *3 x Push button*
- ☐ *3 x 330 Ohm resistor*
- ☐ *Computer with Arduino IDE*

Let's change the brightness of an LED!

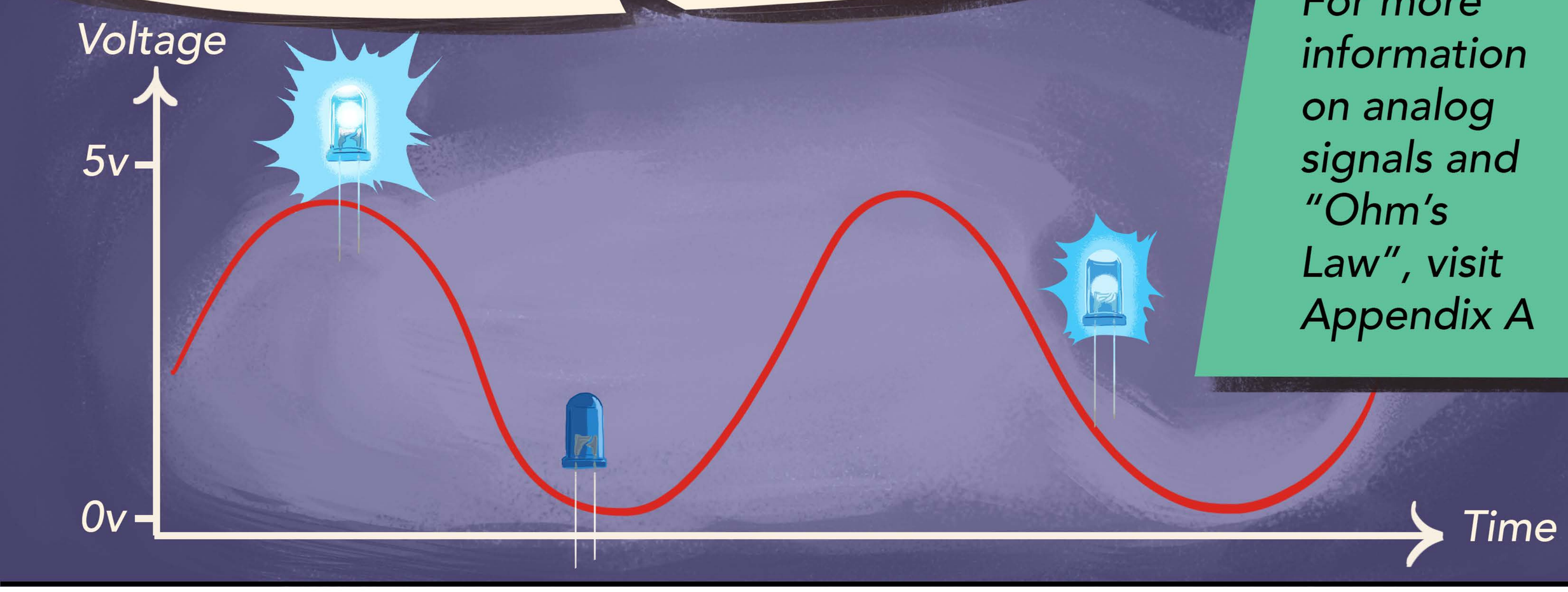
So far, everything we've done is "digital". This means things are on or off, `HIGH` *or* `LOW`*, 0 or 1.*



However, sometimes we need the values in between. For example, a sensor that measures light will have different values in between bright and dark.



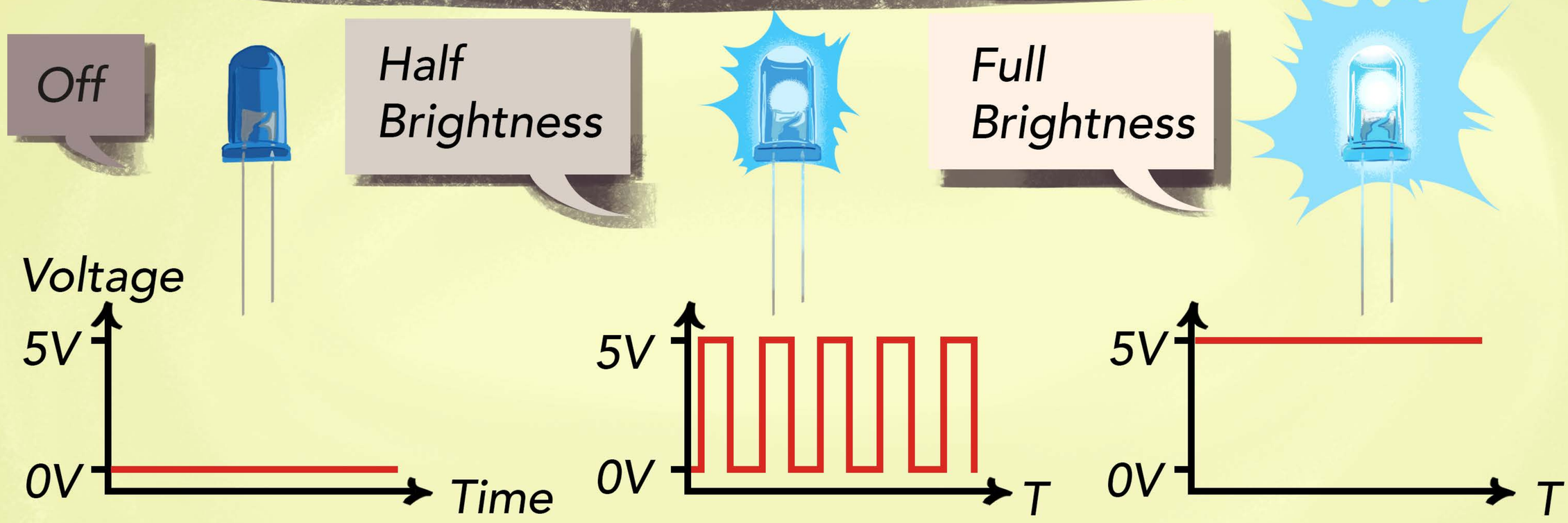
Signals like this are called "Analog" and we can use them to do things like change the brightness of an LED or speed of a motor.



Reference:

For more information on analog signals and "Ohm's Law", visit Appendix A

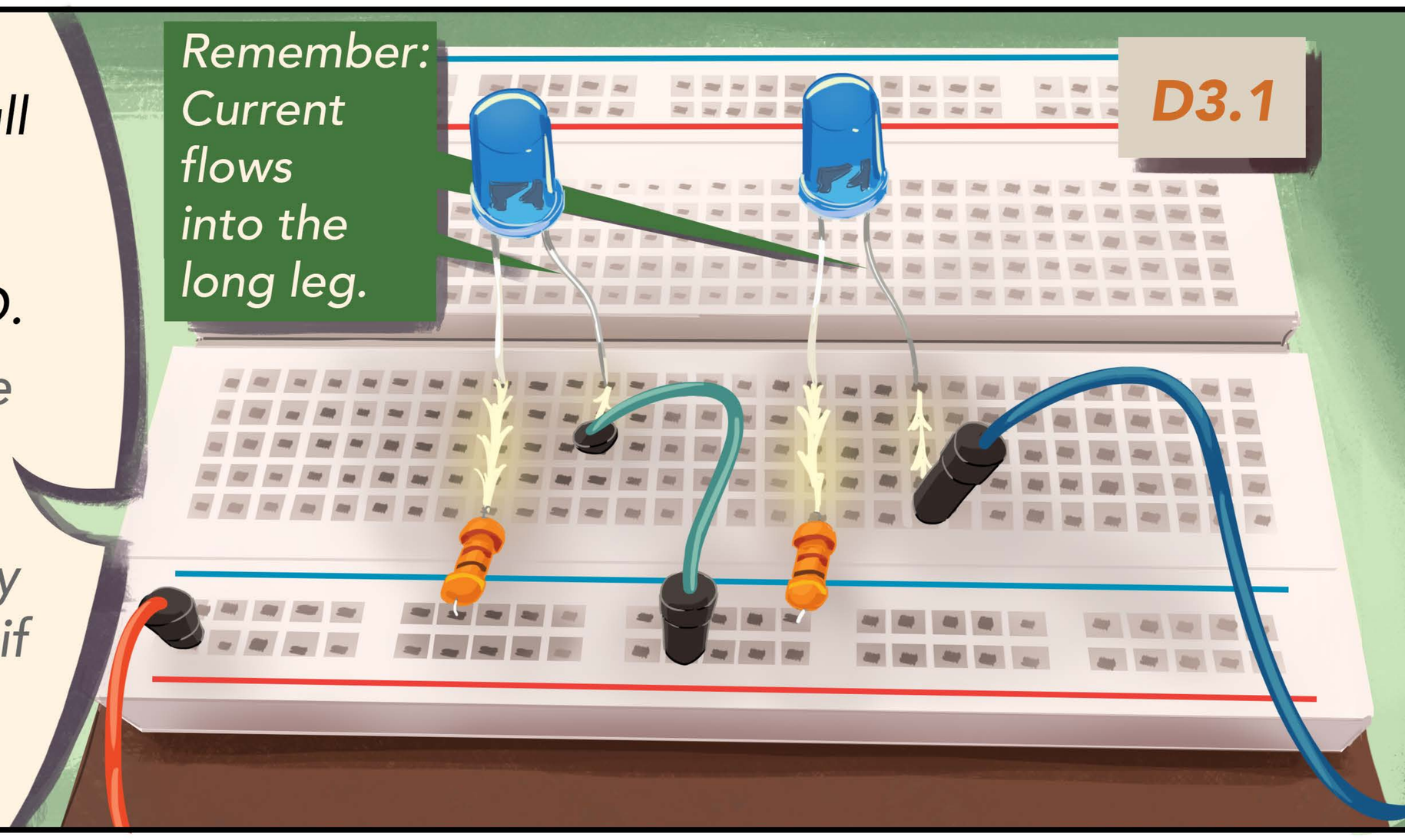
Unfortunately, the UNO doesn't have a way of making true analog voltages. This means we have to simulate it using a method known as Pulse Width Modulation or PWM. This is a way of turning the LED on and off so fast it appears to be dimmer than if it was just on the whole time.



Let's try out this technique. We'll control the brightness of one LED using PWM and have another LED on all the time to compare.

In this circuit, one LED is always at full brightness to compare dimness with the other LED.

Note: Just because your circuit looks different, doesn't mean its wrong! Try to keep going but if it doesn't work check your wiring.



Now, upload this code to your UNO, and see what happens. Notice we're using `delayMicroseconds()` *instead of* `delay()`*, which uses milliseconds.*

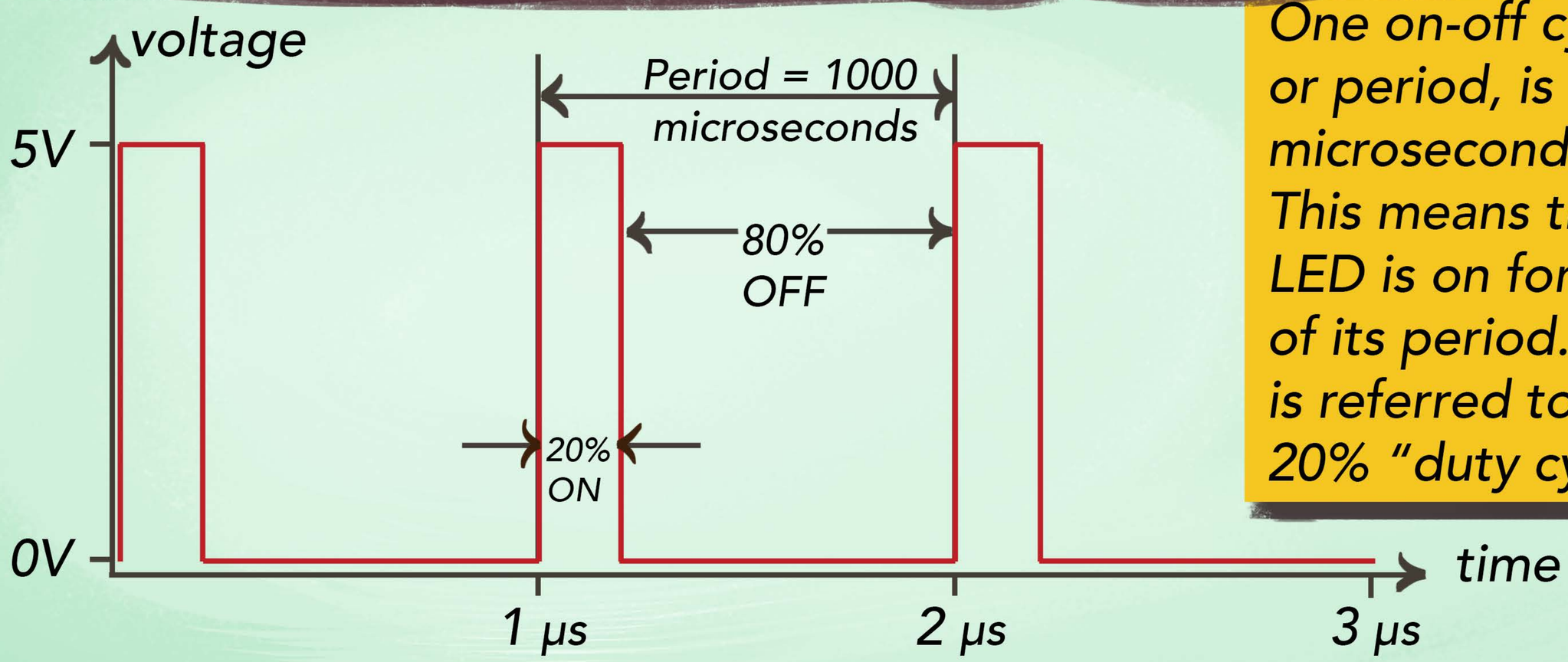
C3.1

```
void setup(){
  pinMode(3, OUTPUT);
}
void loop(){
  digitalWrite(3, HIGH);
  delayMicroseconds(200);
  digitalWrite(3, LOW);
  delayMicroseconds(800);
}
```

Helpful Hint:

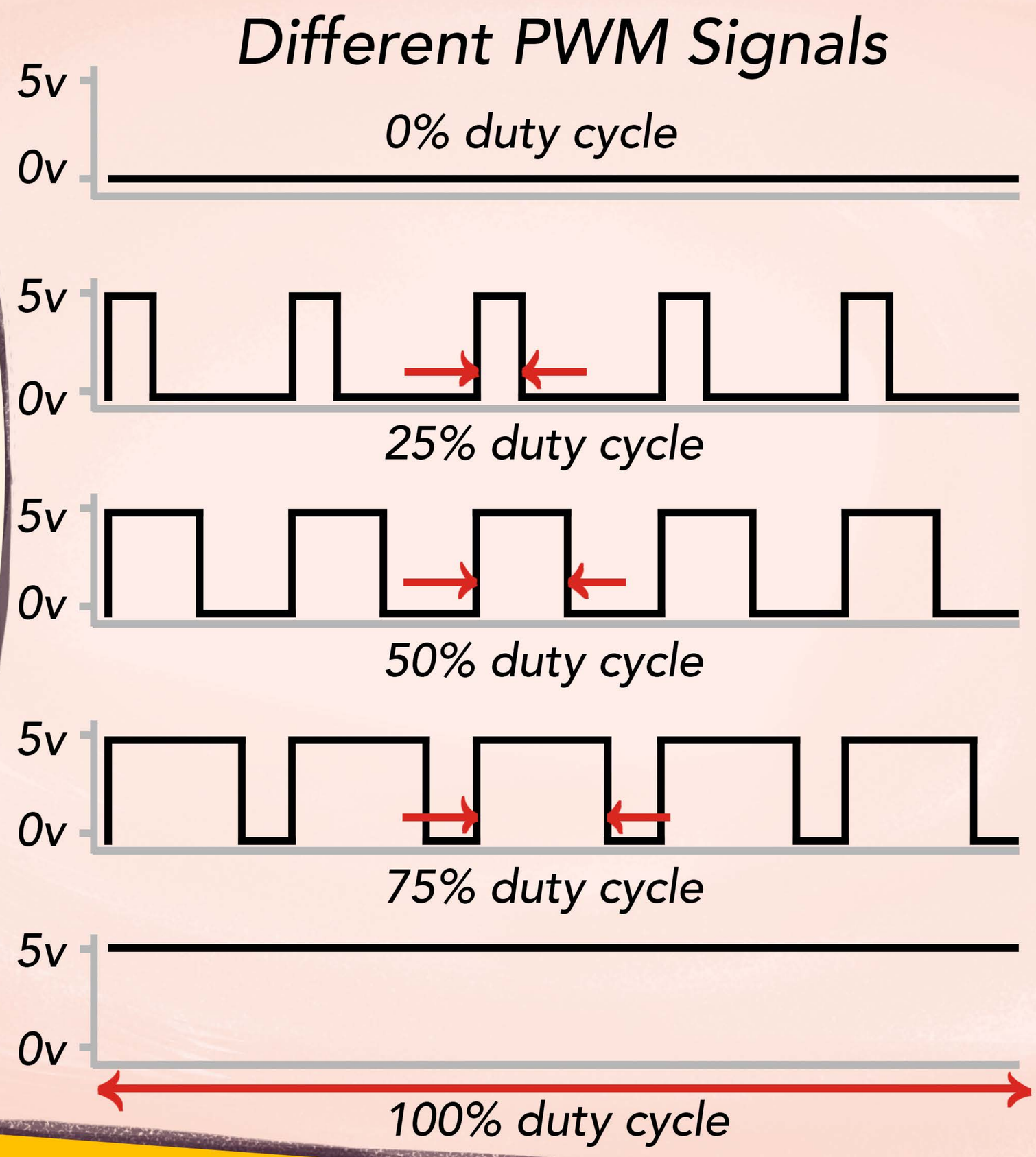
1 second = 1,000 milliseconds (ms)
1 millisecond = 1,000 microseconds (μs)

Notice how we have the LED off for 200 microseconds and on for 800 microseconds. If we could see the voltage on a graph, it would look like this:



One on-off cycle, or period, is 1000 microseconds. This means the LED is on for 20% of its period. This is referred to as a 20% "duty cycle"

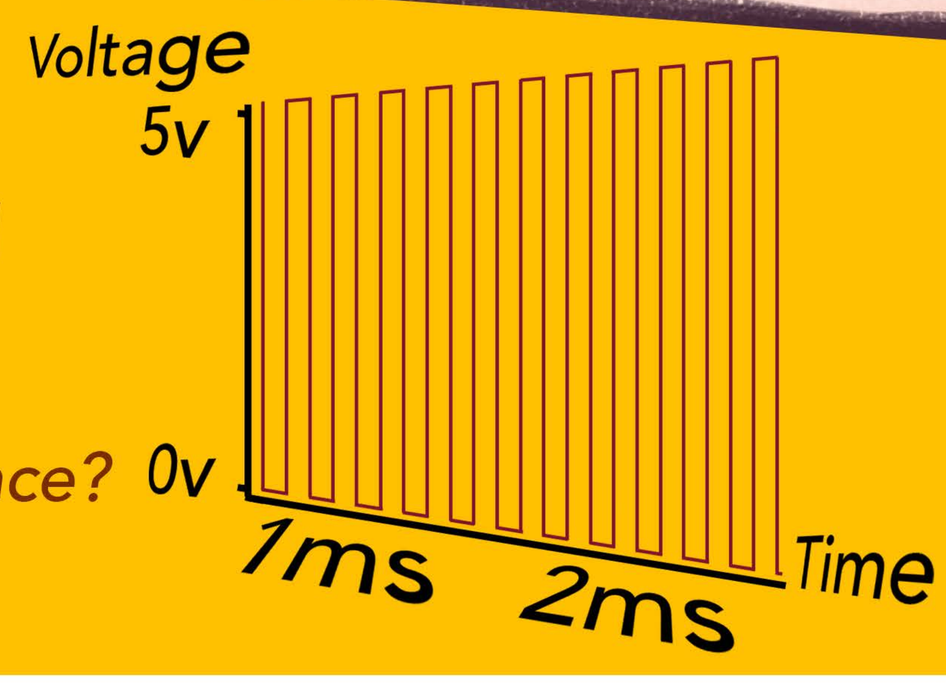
This method uses 5 volt "pulses" of different widths to get different results, in this case, changing the brightness of an LED. That's why it's called "Pulse Width Modulation"

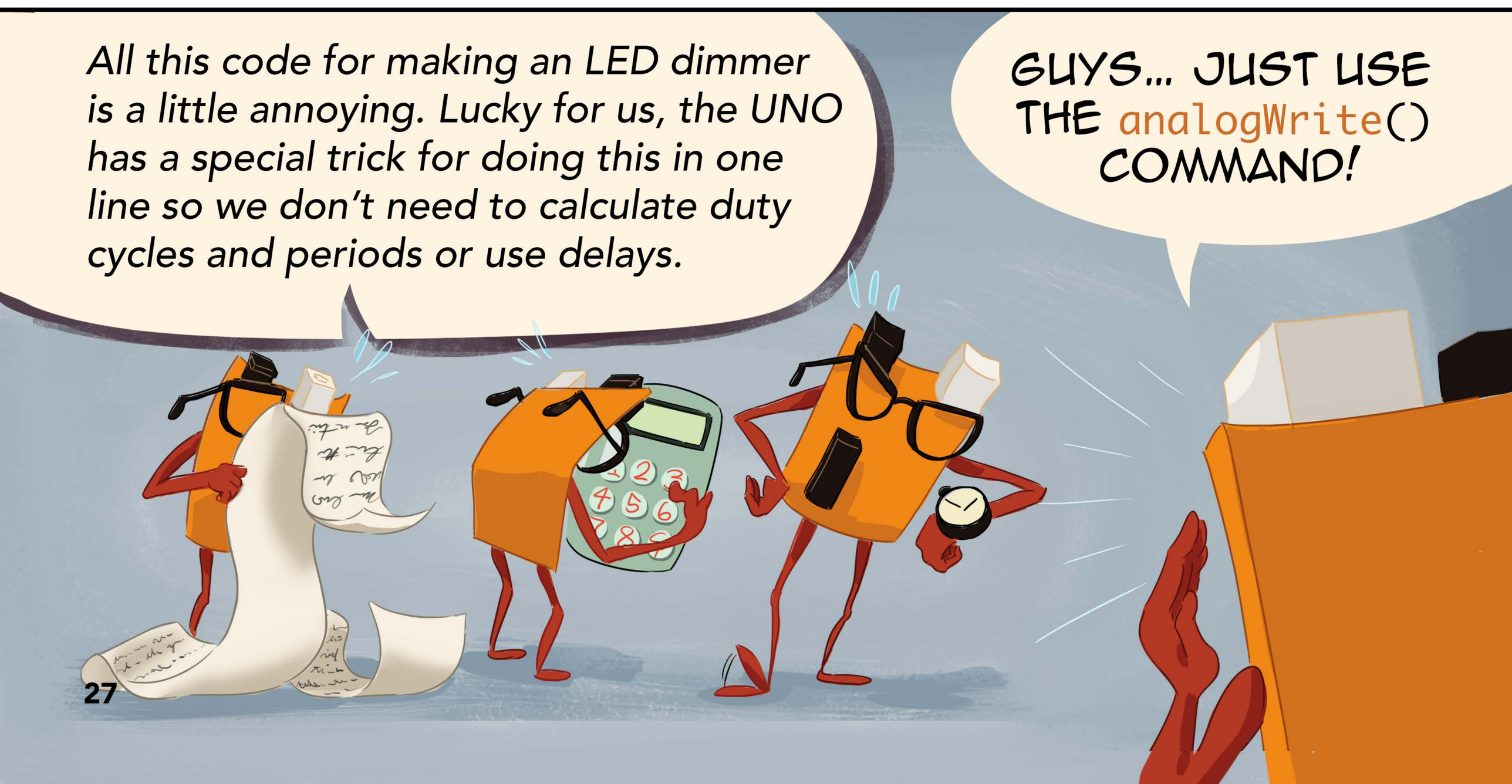


Challenge:

3a. Create a PWM signal with a 2% duty cycle.

3b. Now try the same duty cycle with a period that's 2x as long (2,000 microseconds). Do you notice a difference? why or why not?





C3.2

Try the code below with your same circuit to see how the UNO can produce PWM signals using the `analogWrite()` *function.*

```
void setup(){
  pinMode(3, OUTPUT);
}

void loop(){
  analogWrite(3, 0);
  delay(500);
  analogWrite(3, 127);
  delay(500);
  analogWrite(3, 255);
  delay(500);
}
```

The `analogWrite()` *command takes two numbers or "arguments" and sets the PWM signal on a pin.*

This is the pin we are sending our PWM signal to

```
analogWrite(3, 127);
```

The second argument sets the PWM duty cycle on a scale from 0-255 instead of 0-100% respectively, so 127 is about 50% duty cycle.

Your LED should now be going from off to half, then to full brightness every half a second.

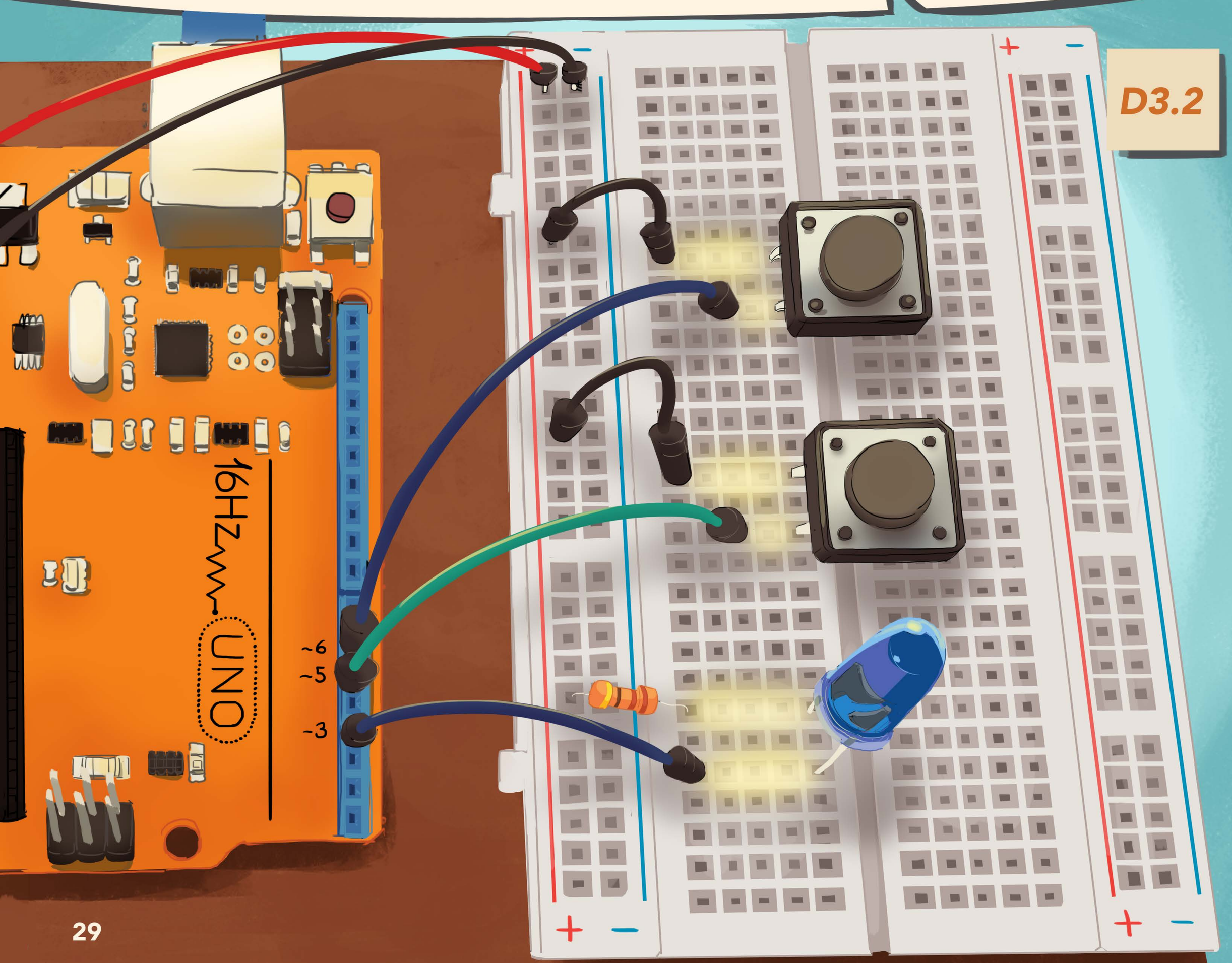
`analogWrite(3, 0)` `analogWrite(3, 127)` `analogWrite(3, 255)`

However, the ability to use `analogWrite()` *is something that not all pins have. Take a close look at all your digital pins and look for the ones that have the "~" symbol. These will let you know which pins can use it.*

Pro-Tip:

Remember that PWM signals have a duty cycle and a period. The `analogWrite()` *command lets you control the duty cycle of the signal by choosing values between 0-255, but the period of the PWM signal it produces on most pins is set to approximately 2040 microseconds. However, Pins 5 and 6 have a period of 1020 microseconds.*

Let's have the UNO increase the brightness of an LED when we press one button, but decrease it when we press the other. Start by building the circuit below.

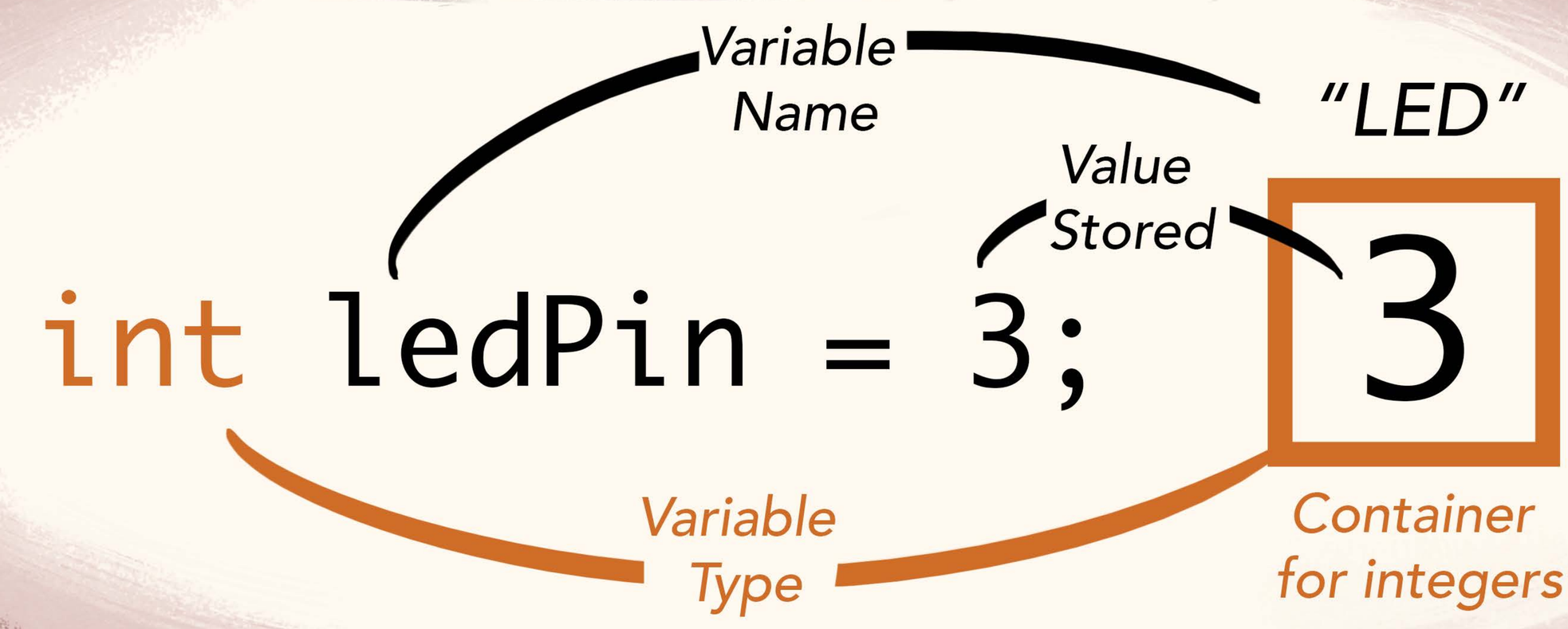


To achieve this effect, we'll need a few more code tricks than we've covered so far. Upload this code and see what happens.

C3.3

```
//These are called variables, they store information
int ledPin = 3;
int brightness = 0;
int button1 = 5;
int button2 = 6;

void setup(){
 pinMode(ledPin, OUTPUT);
 pinMode(button1, INPUT_PULLUP);
 pinMode(button2, INPUT_PULLUP);
}
void loop(){
 //The UNO will use the numbers stored inside the
 //variables instead of the words
 analogWrite(ledPin, brightness);
 //if button1 is pressed, increase brightness
 if(digitalRead(button1) == LOW) && brightness < 255){
   brightness += 1;
 }
 //if button2 is pressed, decrease brightness
 if(digitalRead(button2) == LOW && brightness > 0){
   brightness -= 1;
 }
 //wait a little bit so we don't change too quickly
 delay(10);
}
```

Let's take a look at what the code is doing line by line. The first four lines outside the `setup()` *and* `loop()` *functions are creating "variables". Variables are like containers that store information.*



An "int" variable type can only hold integers (whole numbers), but there are many different types of variables, each designed to store different types of values. We'll look at some of these other data types in later tutorials.

"Bob"

3

```
long Robert = 230,732,985;
float cat = 1.23;
char letter = 'n';
int Bob = 3;  String word = "cheese";
boolean moonLanding = True;
```

"Robert" 230,732,985

"cat" 1.23

"letter" 'n'

"word" "cheese"

"moonLanding" True

Imagine you have a bunch of different components on your breadboard and you decide later you need to move your LED from pin 3 to pin 4. Instead of having to go through your code and changing the pin number from 3 to a 4 everywhere, using a variable allows you to change it in one place. The UNO will know that our variable "ledPin" is now 4, not 3.

```
//These are called variables, they store information
int ledPin = 4;
int brightness = 0;
int button1 = 5;
int button2 = 6;

void setup(){
  pinMode(ledPin, OUTPUT);
  pinMode(button1, INPUT_PULLUP);
  pinMode(button2, INPUT_PULLUP);
}
void loop(){
  //The UNO will use the numbers stored inside the
  //variables instead of the words
  analogWrite(ledPin, brightness);
  //if button1 is pressed, increase brightness
  if (digitalRead(button1) == LOW){
    brightness += 1;
```

Change it here and it changes everywhere!

Variables also let us use convenient names like "ledPin" or "button1" instead of having to remember "3" or "5".

We can even change the values stored inside the variable as the program runs. Take a look at what's happening with the "brightness" variable in our code.

Creates variable and starts it out at "0"

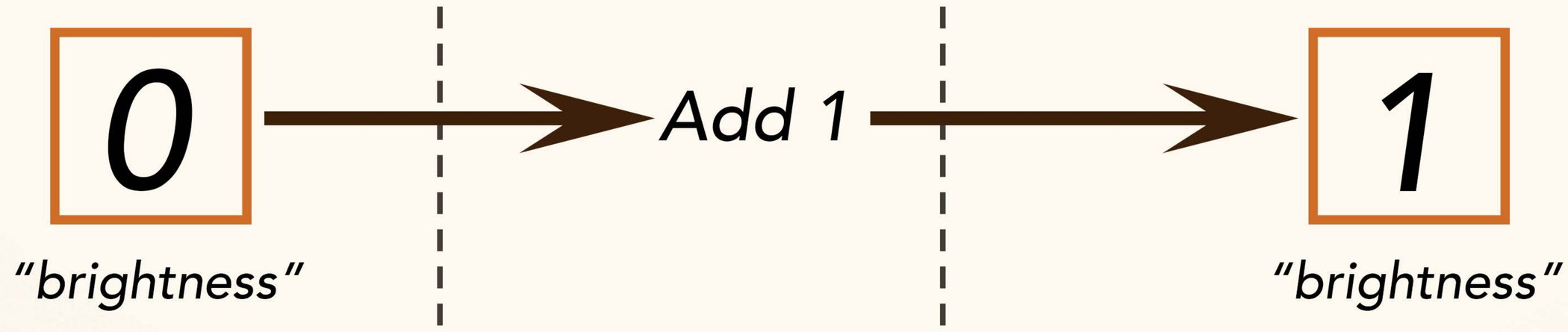
Adds "1" to the current value inside "brightness"

The variable now contains the updated value!

```
int brightness = 0;
```

```
brightness += 1;
```

```
analogWrite(ledPin, brightness);
```



Pro-Tip:

These two statements are actually identical: "`brightness += 1;`" and "`brightness = brightness + 1;`" The operator "+=" (and also "-=") are just shorthand that make writing code a little easier.

Now that we know what variables are, lets look at our code step by step.

```
//These are called variables, they store information
int ledPin = 3;
int brightness = 0;
int button1 = 5;
int button2 = 6;

void setup(){
 pinMode(ledPin, OUTPUT);
 pinMode(button1, INPUT_PULLUP);
 pinMode(button2, INPUT_PULLUP);
}
```

You can't put something in a container that doesn't exist, so we need to "declare" (create) our variables before we use them. This is usually done all the way at the top, before `void setup()`.

Here we configure our pins using `pinMode()`, but notice how now we can use the variables instead of the numbers themselves. Isn't it easier to read?

The `void loop()` is trickier. Let's see how changing our variables and using the `analogWrite()` command gets us our fading effect.

This is where we set the duty cycle of the "`ledPin`". It always sets the value to whatever we make "brightness" to be.

The "&&" is a special operator that lets us check if two different conditions are both true.

```
void loop(){
  //The UNO will use the numbers stored inside the
  //variables instead of the words
  analogWrite(ledPin, brightness);
  //if button1 is pressed, increase brightness
  if (digitalRead(button1) == LOW && brightness < 255){
    brightness += 1;
  }
  //if button2 is pressed, decrease brightness
  if (digitalRead(button2) == LOW && brightness > 0){
    brightness -= 1;
  }
  //wait a little bit so we don't change too quickly
  delay(10);
}
```

In this case, we want to increase brightness by one IF the first button is pressed AND ("&&") "brightness" is less than (255)"

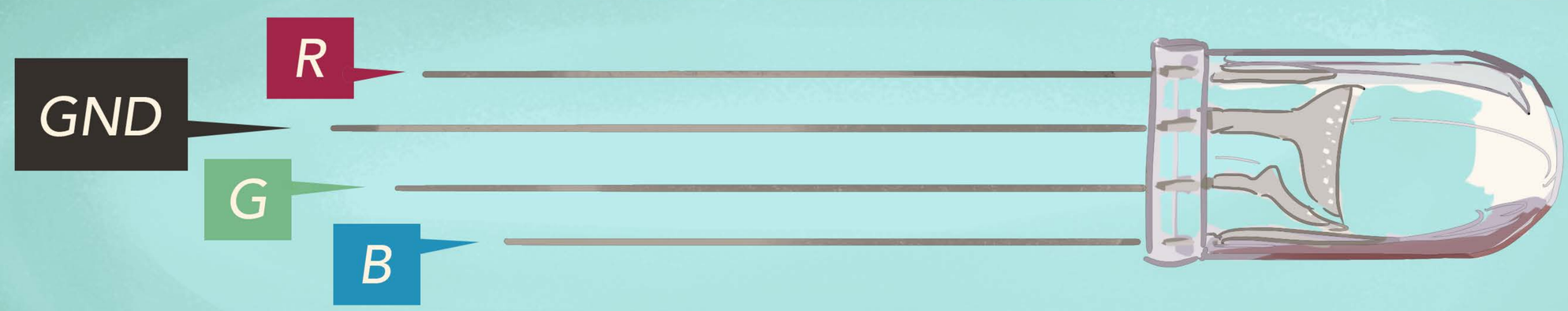
This time we want to decrease "brightness" by one IF the second button is pressed AND ("&&") "brightness" is more than the minimum brightness (0)

Helpful Hint:

Remember that the `analogWrite()` function can only take certain values. That's why we need the "&&" conditions to keep "brightness" between 0 and 255. If the values were to go above 255 or below 0, `analogWrite()` would interpret them as wrapping around the other side. This might sound confusing, but try replacing the two if statements above with this code and you'll see.

```
if (digitalRead(button1) == LOW) {
  brightness += 1;
}
if (digitalRead(button2) == LOW) {
  brightness -= 1;
}
```

Now let's use the `analogWrite()` *command to control the 3 different colors of an RGB LED. An RGB LED has three different diodes inside one package. Since the LED in your kit is a "common cathode" the 4 pins it has are red, green, blue and ground. If it was "common anode" it would be red, green, blue and positive voltage.*

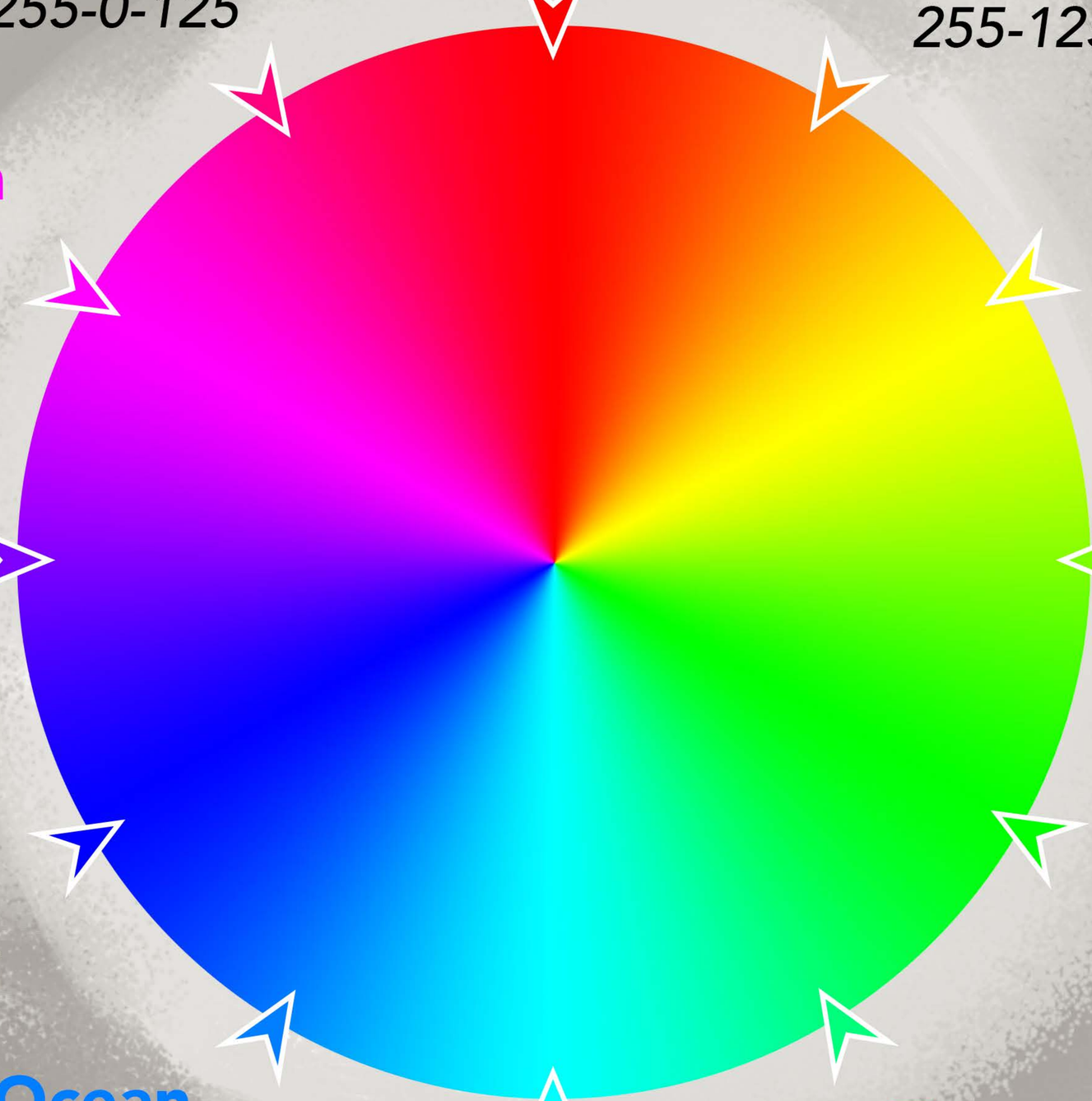


Helpful Hint:

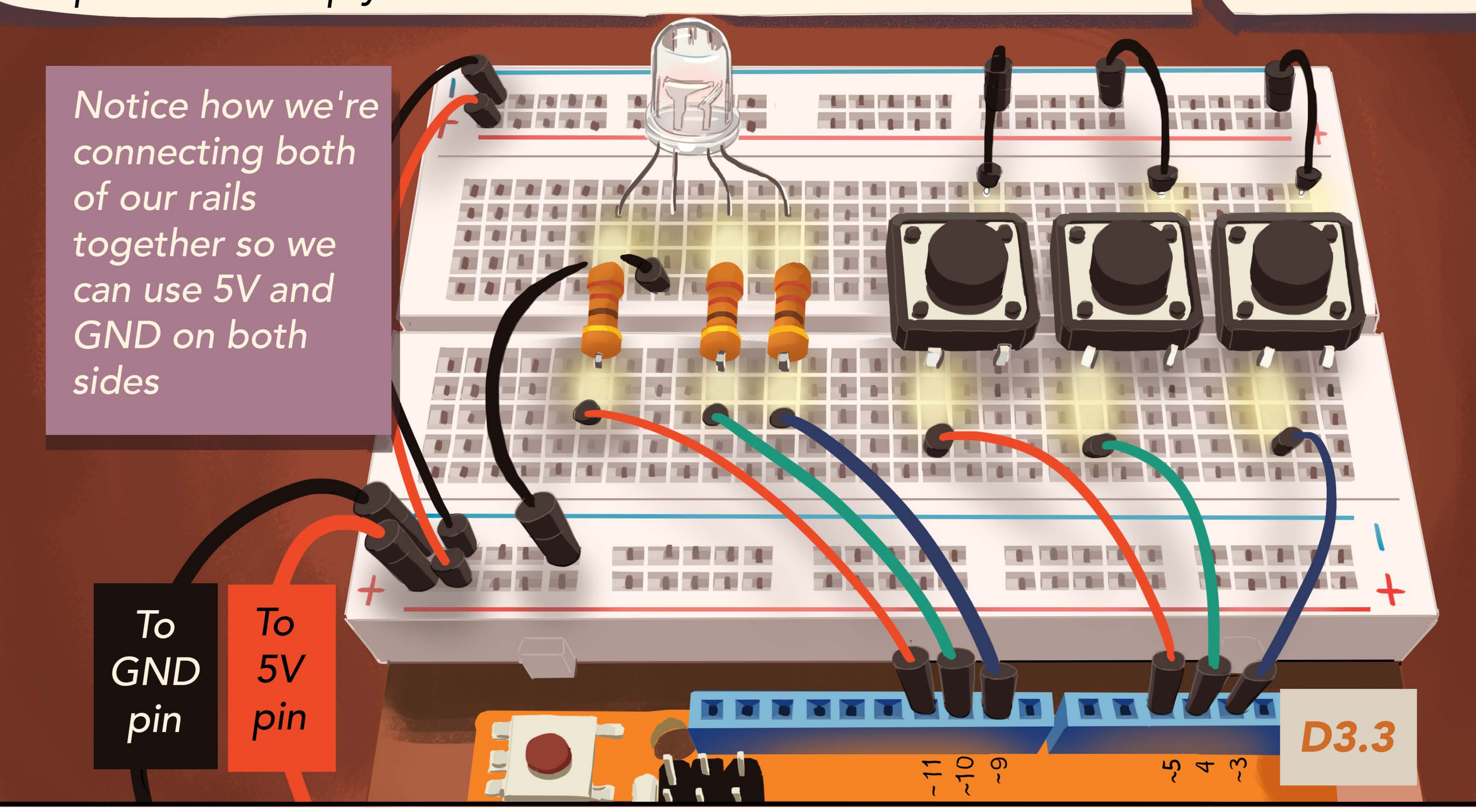
If you ever forget which pin corresponds to which color, you can always just connect the longest pin to ground and power one diode at a time to see which pin is which color.

RGB LEDs are great because any color in the spectrum can be made by combining different amounts of red, green and blue.

Color Wheel:



We're going to use 3 buttons to control the 3 diodes in our RGB LED. Let's make the UNO pick a different color when each of our 3 buttons are pressed. Set up your breadboard like so.



Let's make each button change the LED to a different color. Upload this new code and see what happens. Button 1 will make yellow, 2 will make cyan, 3 will make magenta.

C3.4

```
//Look at all these variables!
int redPin = 11;
int greenPin = 10;
int bluePin = 9;

int button1 = 5;
int button2 = 4;
int button3 = 3;

int redValue = 0;
int greenValue = 0;
int blueValue = 0;

void setup(){
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  pinMode(button1, INPUT_PULLUP);
  pinMode(button2, INPUT_PULLUP);
  pinMode(button3, INPUT_PULLUP);
}
```

These variables will store our pin numbers

These variables will store the values we want our LEDs to be

Continued on next page

Continued from last page.

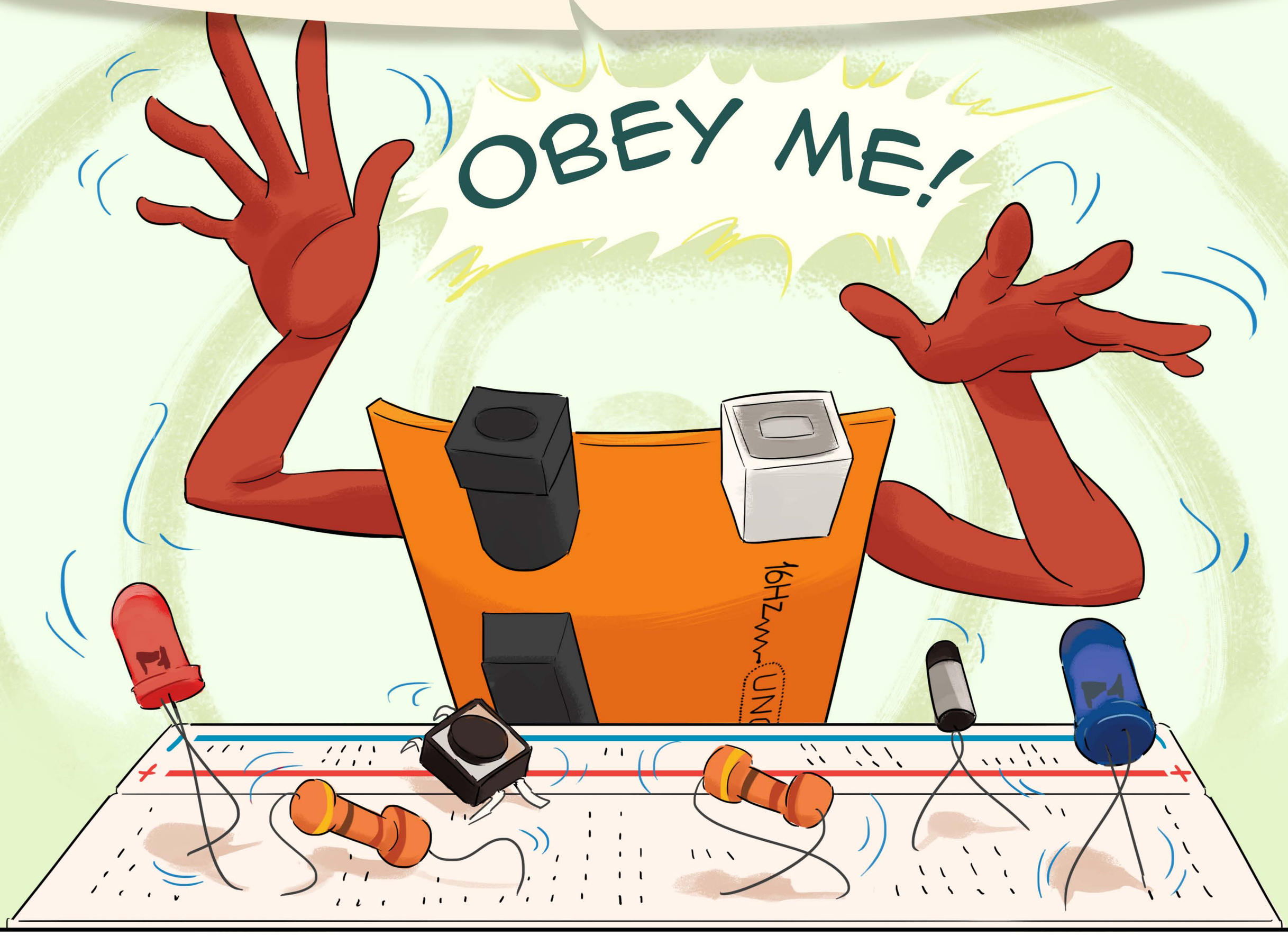

```
void loop(){

  //Set the LEDs to the right values
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);

  //if button1 is pressed, make a yellow!
  if (digitalRead(button1) == LOW){
    redValue = 255;
    greenValue = 255;
    blueValue = 0;
  }
  //if button2 is pressed, make cyan!
  if (digitalRead(button2) == LOW){
    redValue = 0;
    greenValue = 255;
    blueValue = 255;
  }
  //if button3 is pressed, make magenta!
  if (digitalRead(button3) == LOW){
    redValue = 255;
    greenValue = 0;
    blueValue = 255;
  }
  //wait a little bit
  delay(10);
}
```

Notice how the "==" is asking if the button is LOW while the "=" is telling the variable to be a specific value.

Challenge:

3c. *Use the color wheel and pick a different color to use with each of the 3 buttons.*

The great thing about microcontrollers like the UNO, is that you can make one circuit do lots of different things just by changing the code.



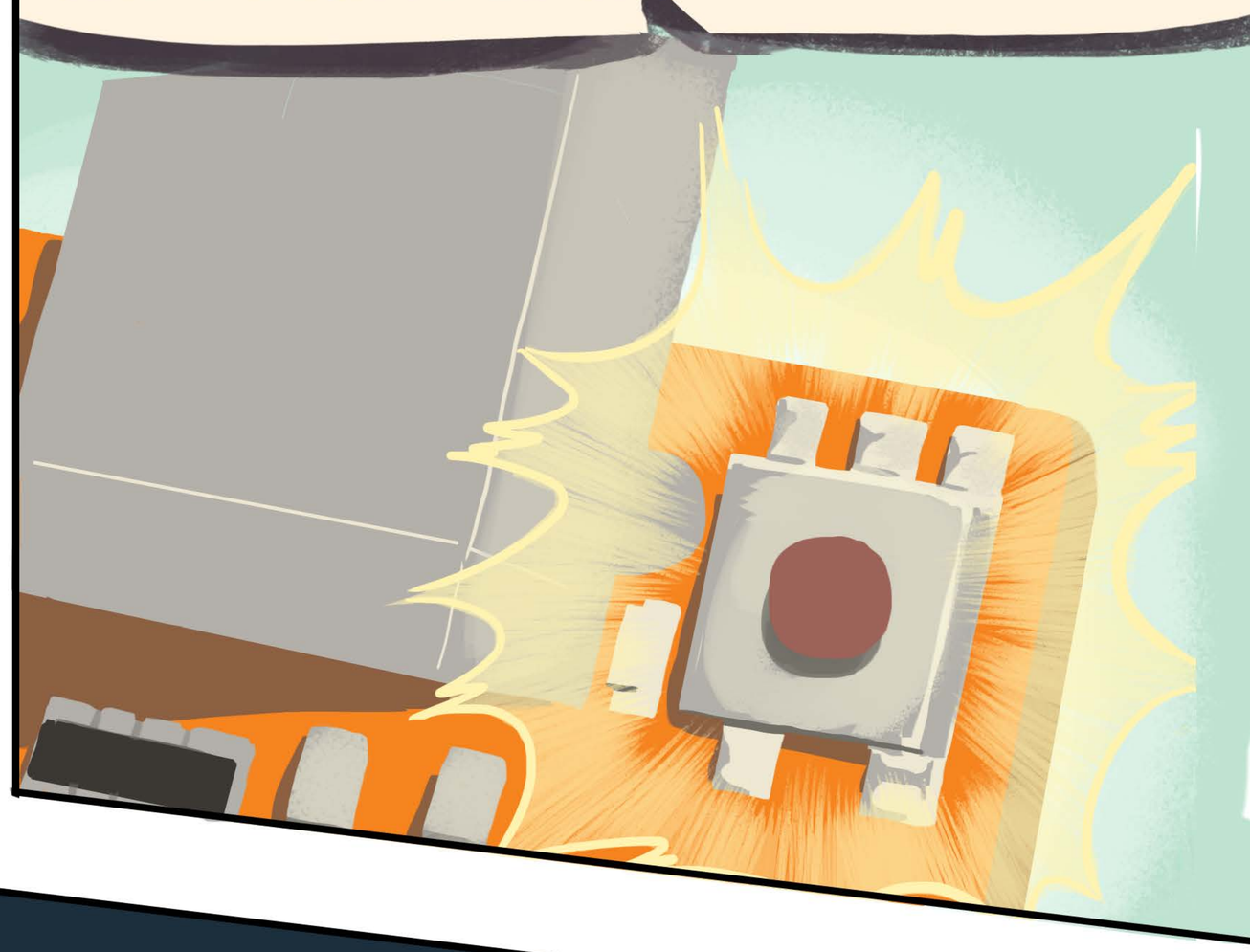
Wouldn't it be great if we could change the colors of the LED fluidly to get any color we want without having to upload new code every time?
Try replacing the three if statements in your code with these new ones.

C3.5

```
//if button1 is pressed, increase red value
 if(digitalRead(button1) == LOW && redValue < 255) {
   redValue += 1;
 }
 //if button2 is pressed, increase green value
 if(digitalRead(button2) == LOW && greenValue < 255) {
   greenValue += 1;
 }
 //if button2 is pressed, increase blue value
 if(digitalRead(button3) == LOW && blueValue < 255) {
   blueValue += 1;
 }
```

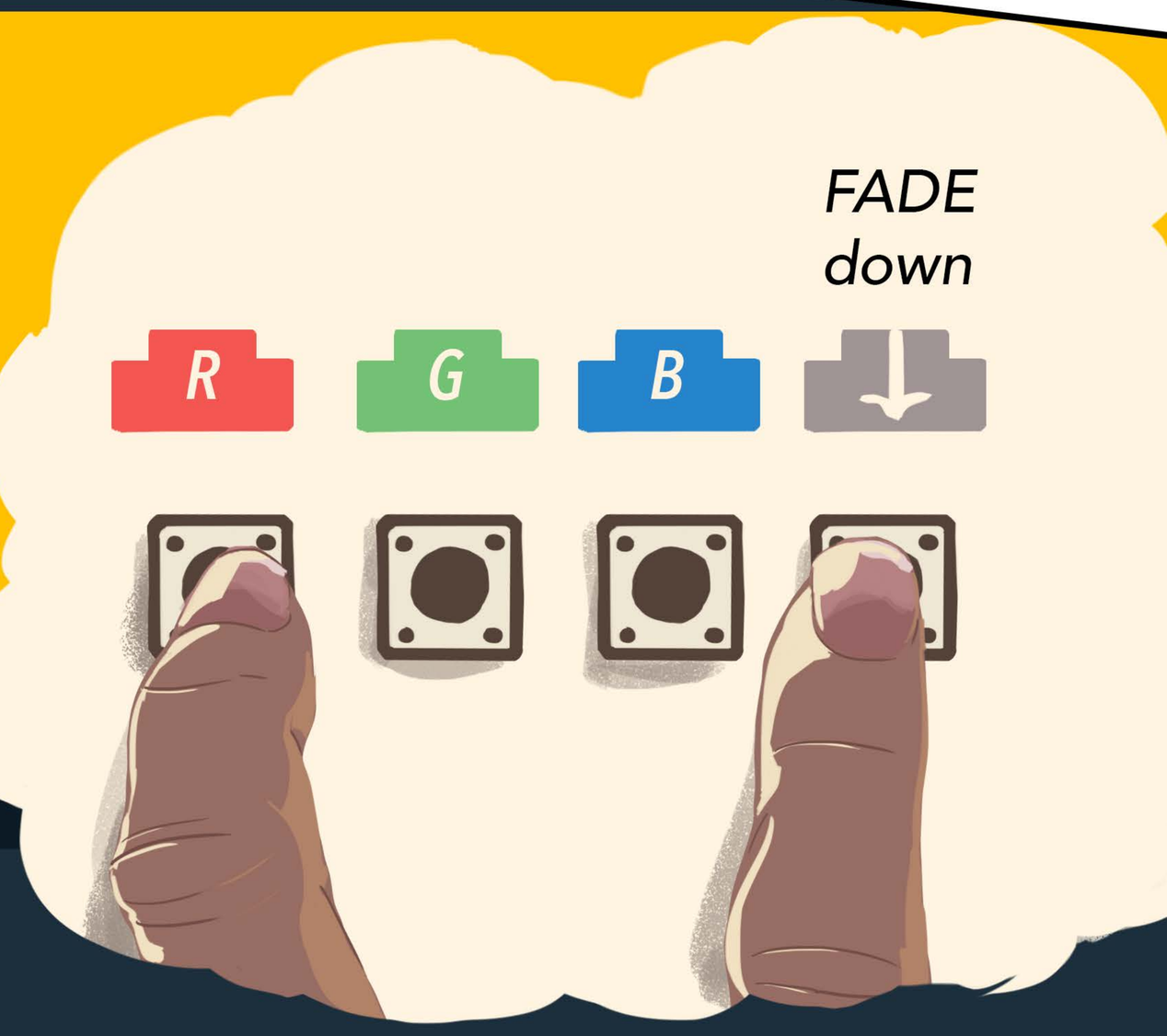
You can now control how much red, green and blue the LED will have by holding each of the buttons down to increase the individual R,G and B values.

If you look on your UNO next to the USB jack, you'll find the reset button. It tells your UNO to stop what it's doing and start over from the first line of the code currently on it. In this case, we can use it to reset our colors back to zero.



Challenge:

3d. *Have the UNO fade colors up like before, but now add a 4th button to make a color fade down if it is pressed at the same time as that respective colors' fade up button. Hint: you can have if statements inside of if statements*



3e. *Have the UNO produce a random color when a button is pressed. Hint: the "*`random(MIN, MAX)`*" command will give you a random number between whatever numbers you replace "*`MIN`*" and "*`MAX`*" with.*

TUTORIAL 4:
ANALOG READ

16Hertz UNO, breadboard, and plate

RGB LED

LET'S LEARN ABOUT:

Using sensors to measure analog values like light and temperature

3 330 Ohm Resistors

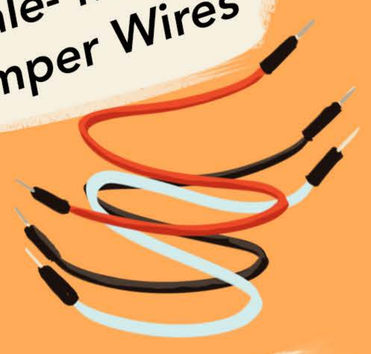
MATERIALS:

- ☐ 16Hertz UNO + Breadboard
- ☐ USB cable
- ☐ Breadboard
- ☐ Male/male Jumper wires
- ☐ RGB LED
- ☐ 3 x 330 ohm Resistors
- ☐ 10 kilo-ohm resistor
- ☐ Potentiometer
- ☐ Photoresistor
- ☐ Thermistor
- ☐ Digital Multimeter
- ☐ Computer with Arduino IDE

2x Alligator Clips

Arduino IDE

Male- Male Jumper Wires



1K, 2K, and 10K-ohm resistors

Potentiometer

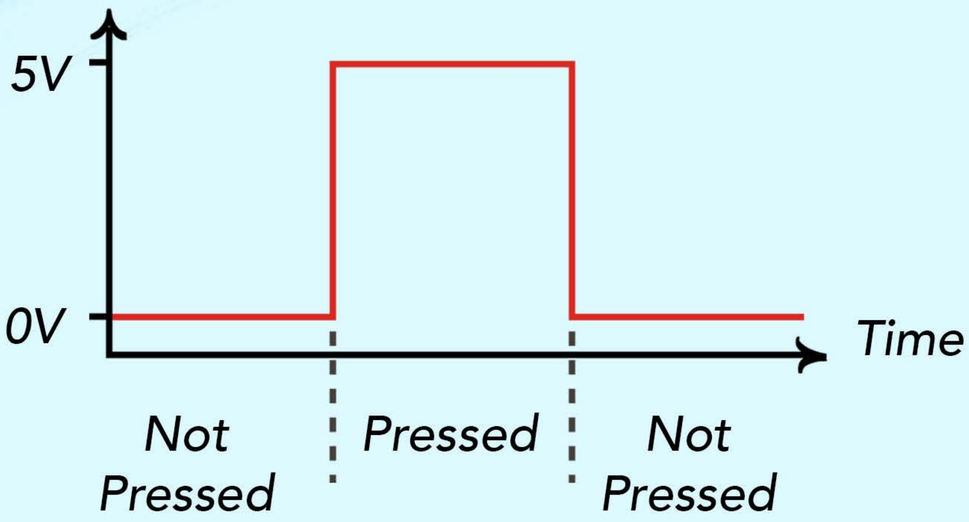
Digital Multimeter

Photoresistor

Thermistor

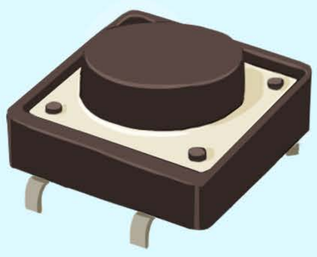
Let's use some sensors!

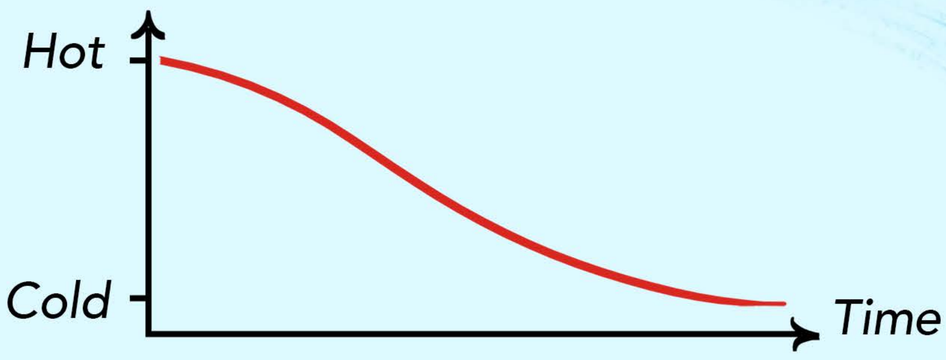
So far, we've used digital inputs from buttons to read what is happening outside the UNO. However, sometimes we want to measure quantities like light and temperature that can't be expressed simply as "on" of "off". For these cases, we use analog sensors.



Digital Sensor

Momentary Switch (button)





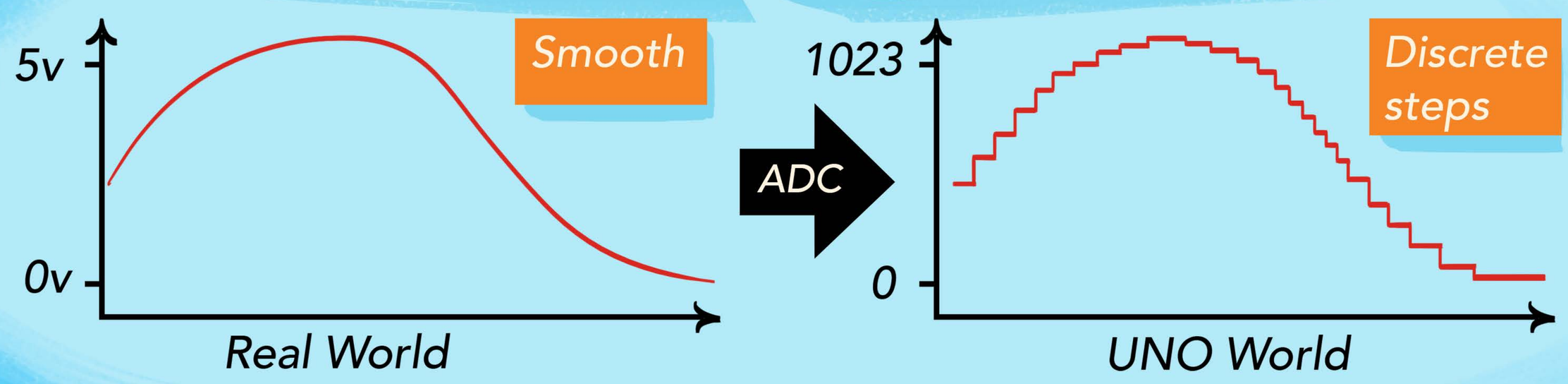
Coffee Temperature

Analog Sensor

Thermistor (temperature sensor)

To read an analog sensor, we need to use the analog pins. These pins are connected to special circuitry that can convert incoming voltages from 0-5 volts into numbers between 0-1023 respectively. We can then read them with the Serial Monitor and use them in our code.

In the real world analog signals are continuous, meaning they have infinitely many steps. However, the circuitry on the UNO has limited resolution. The "Analog to Digital Converter" or ADC can divide the 0 to 5 volt voltage range into 1024 discrete steps. This means each step is about 0.0049 volts, or 4.9 millivolts, so your UNO can read sensors very accurately!



Pro-Tip:

The ADC uses 0-5 volts as the "reference" range by default and this is great for almost all projects including the ones in this guide. However, if you need more resolution per step than 4.9 millivolts, look into the UNO's built-in `analogReference()` *function.*

Our analog sensors are "variable resistors". They have the ability to change their resistance based on an external factor. This makes them great at creating different voltages we can read using our analog pins.

Photoresistor - *changes resistance with ambient light*

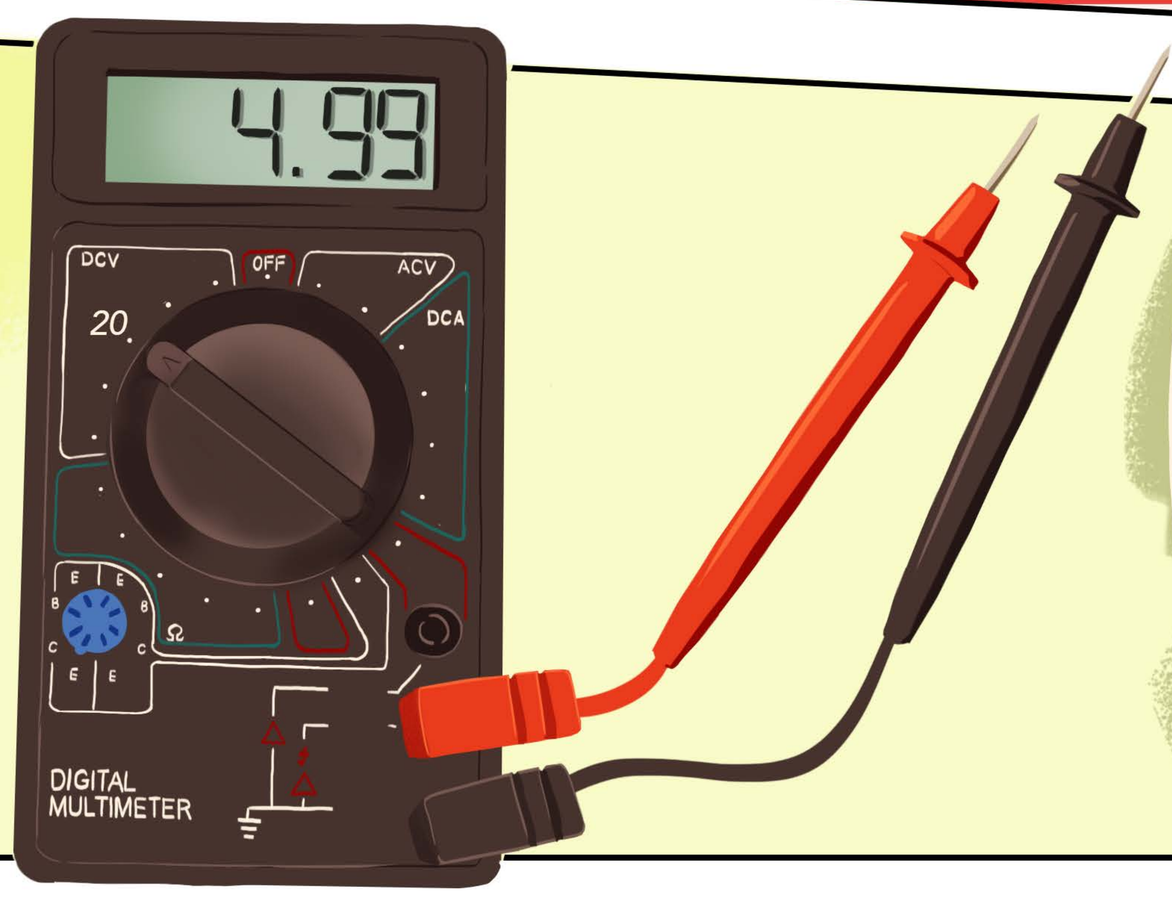
Potentiometer - *changes resistance when turned*

Thermistor - *changes resistance with temperature*

Let's take a closer look at how resistors can affect voltage. Add a 1000 ohm resistor between the 5V and ground rails and use your multimeter to measure the voltage across this resistor.

To 5 volts

To GND



In a closed circuit, all the voltage has to be used up by the time it goes from power to ground. If there is only one component in the circuit, it uses up all the voltage. The voltage used up across a component is called "voltage drop".

Let's see what happens when we share the voltage between two components. Add a 1 kilo-ohm and 2 kilo-ohm resistor to your breadboard between the 5V and ground rails as shown here.

To GND

To 5 volts

2K ohm

1K ohm

Use your multimeter and measure the voltage across each resistor individually. What values do you get? What do you get if you add both the values?

You get 5 volts! This setup is known as a "voltage divider". Voltage dividers essentially let us "divide" any input voltage we have into an output voltage we want by using up a portion of the input across each of the resistors. The voltage drop across each resistor depends on the ratio between that resistor and the total resistance in the circuit. We can even calculate what it will be using the handy voltage divider equation:

The resistance of the section being measured

The total voltage across the circuit

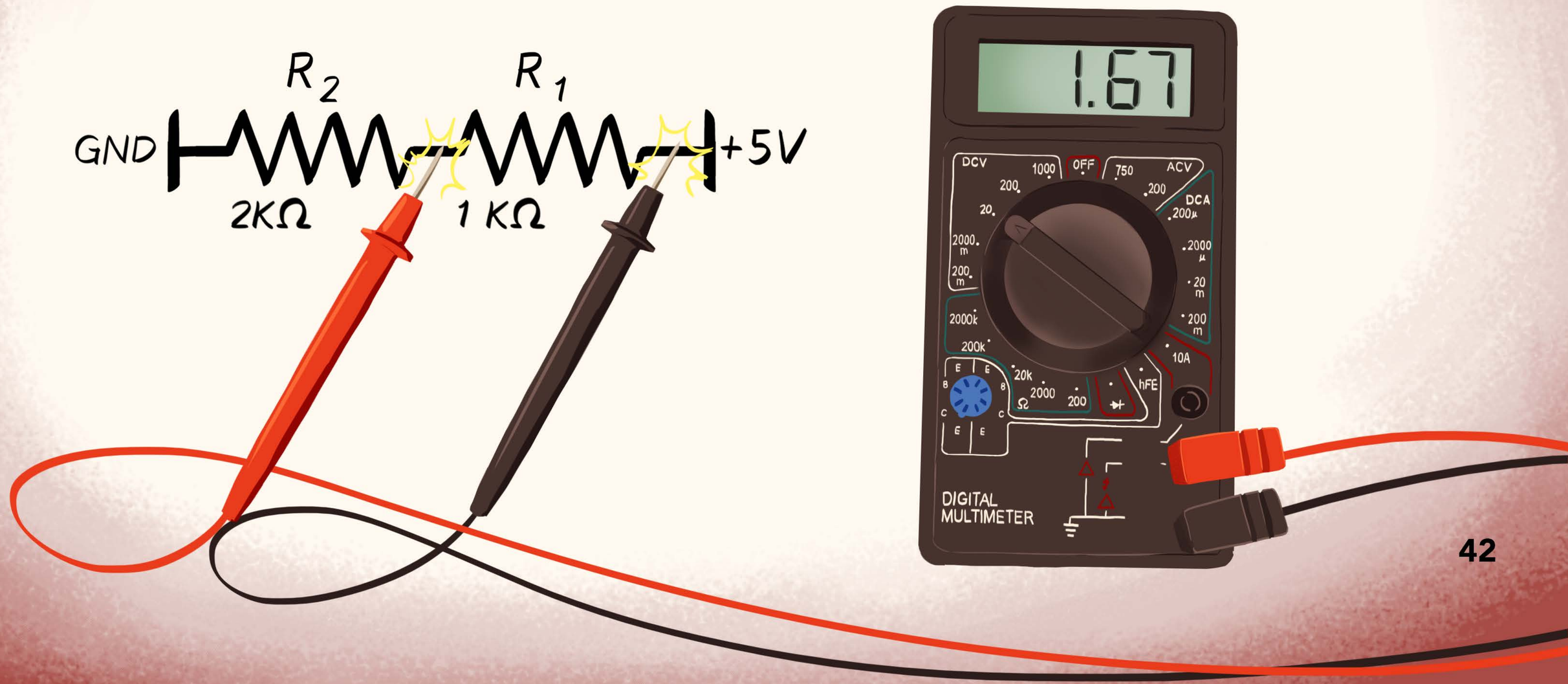
$$V_{Output} = \frac{R_1}{R_1 + R_2} \times V_{Input}$$

The voltage across the section being measured

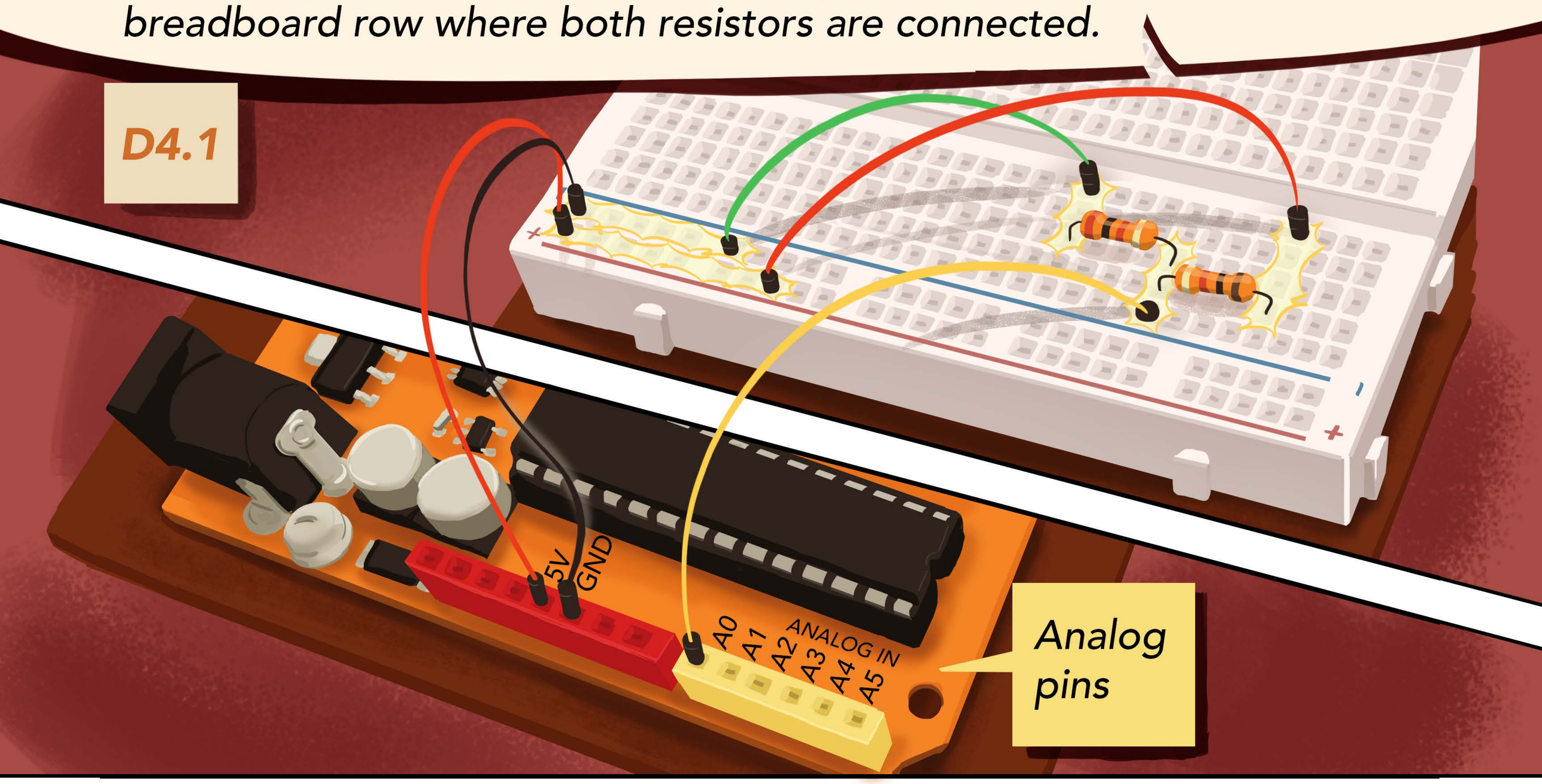
The total resistance in the circuit

In this case, we get about 1.67 volts when measuring the voltage across the 1K ohm resistor. Let's see what it looks like when we use the UNO to measure this voltage instead of a multimeter.

$$V_{Output} = \frac{1000 \text{ ohms}}{3000 \text{ ohms}} \times 5 \text{ volts} = 1.67 \text{ volts}$$



Use a jumper wire to connect Analog pin 0 on your UNO to the breadboard row where both resistors are connected.



To see what values the UNO is reading from the ADC, upload this code to your UNO and open up the serial monitor.

```
void setup(){
  Serial.begin(9600);

}

void loop(){
  Serial.println(analogRead(0));
  delay(50);

}
```

Notice how we don't need to set the `pinMode()`*. This is because analog pins are always inputs.*

C4.1

Helpful Hint:

Voltage, like height, needs two points to be defined, a "top" and a "bottom". A person is 6 feet tall measured from the ground. In order to read voltage, we need to compare it to a reference, which is usually GND. This is why the multimeter needs two connections to measure voltage. The UNO only needs us to connect the measurement point to analog pin 0 because it has its own internal "reference to ground".

/dev/tty.usbmod

```
340
334
335
336
```

The value you see in the serial monitor should be somewhere around 340. This is about 1/3 of the 1024 total steps, just like the way the multimeter measured 1.67 volts which is 1/3 of 5 volts.

Take the jumper wire out from between the resistors and connect analog pin 0 directly to ground.

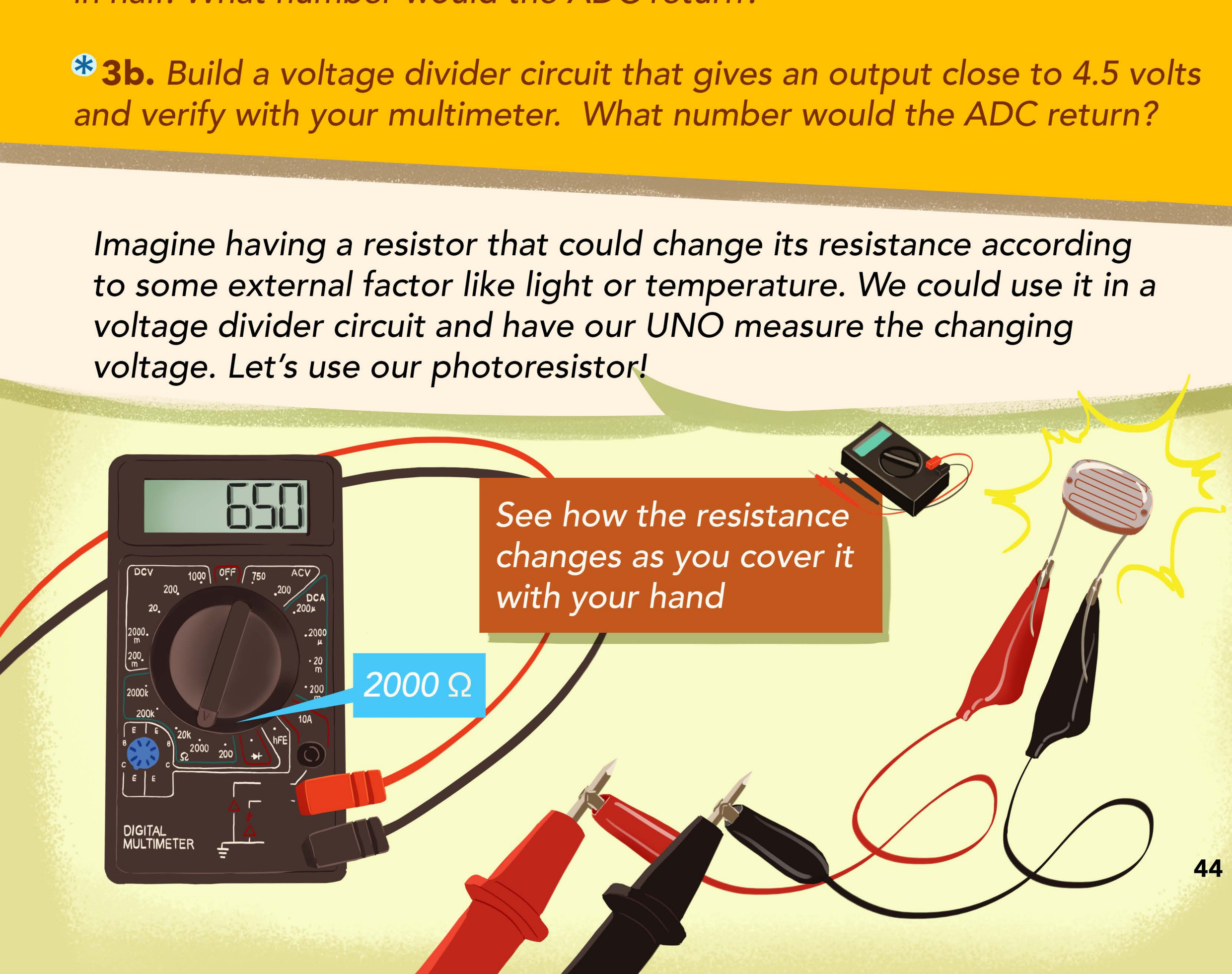
What reading do you expect to see in the serial monitor? What if you connected it to 5V instead?

Challenge:

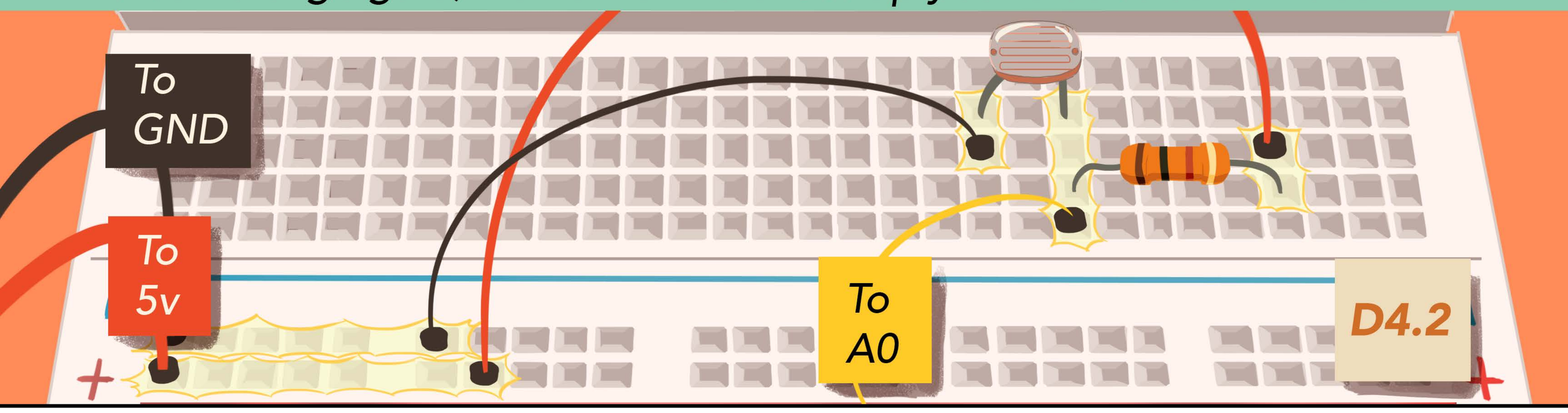
3a. *Find the combination of resistors that would split the voltage in half. What number would the ADC return?*

3b. *Build a voltage divider circuit that gives an output close to 4.5 volts and verify with your multimeter. What number would the ADC return?*

Imagine having a resistor that could change its resistance according to some external factor like light or temperature. We could use it in a voltage divider circuit and have our UNO measure the changing voltage. Let's use our photoresistor!

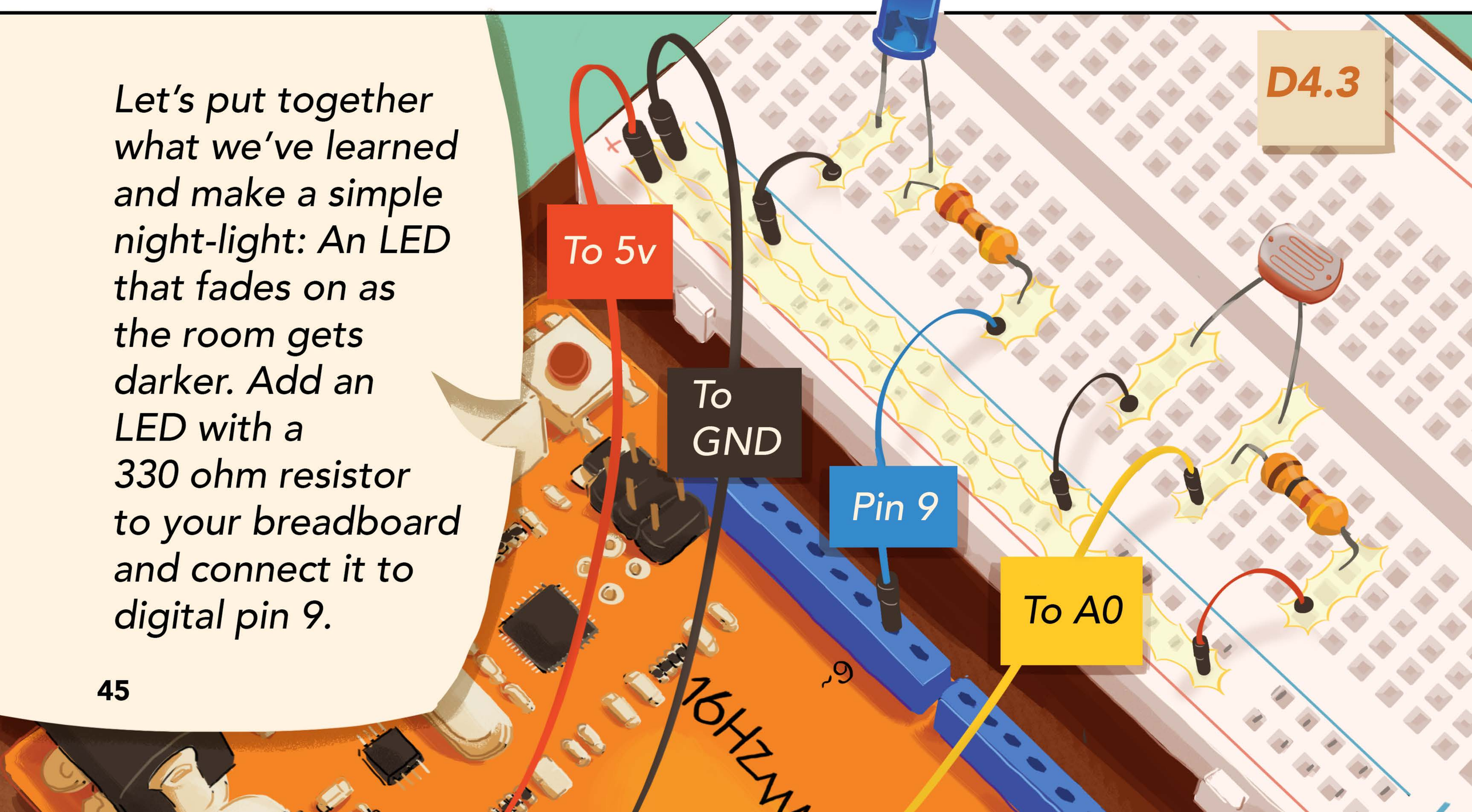


In order to get the most out of this photoresistor we need to pair it with a fixed resistor of an appropriate value. A 10K resistor (brown-black-orange-gold) works well here. Set up your breadboard like so.



Open up the serial monitor and see how the values are affected as you change the light falling on the sensor. Write down the approximate values you see for the darkest and brightest conditions you find. We'll use them later!

Let's put together what we've learned and make a simple night-light: An LED that fades on as the room gets darker. Add an LED with a 330 ohm resistor to your breadboard and connect it to digital pin 9.

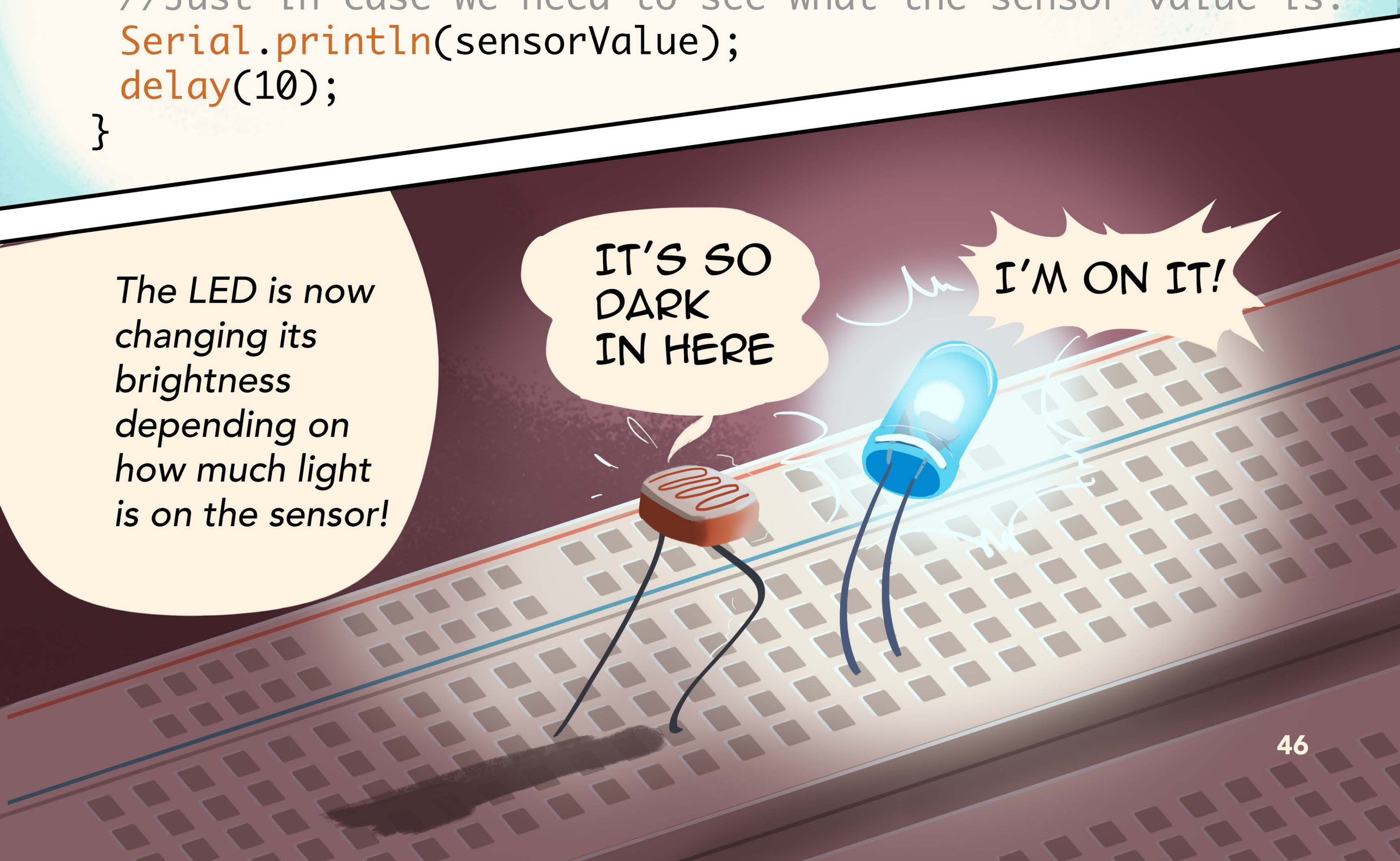


In order to change the brightness of our LED using the value of the photoresistor we'll need to get a bit creative. Upload this code to your UNO.

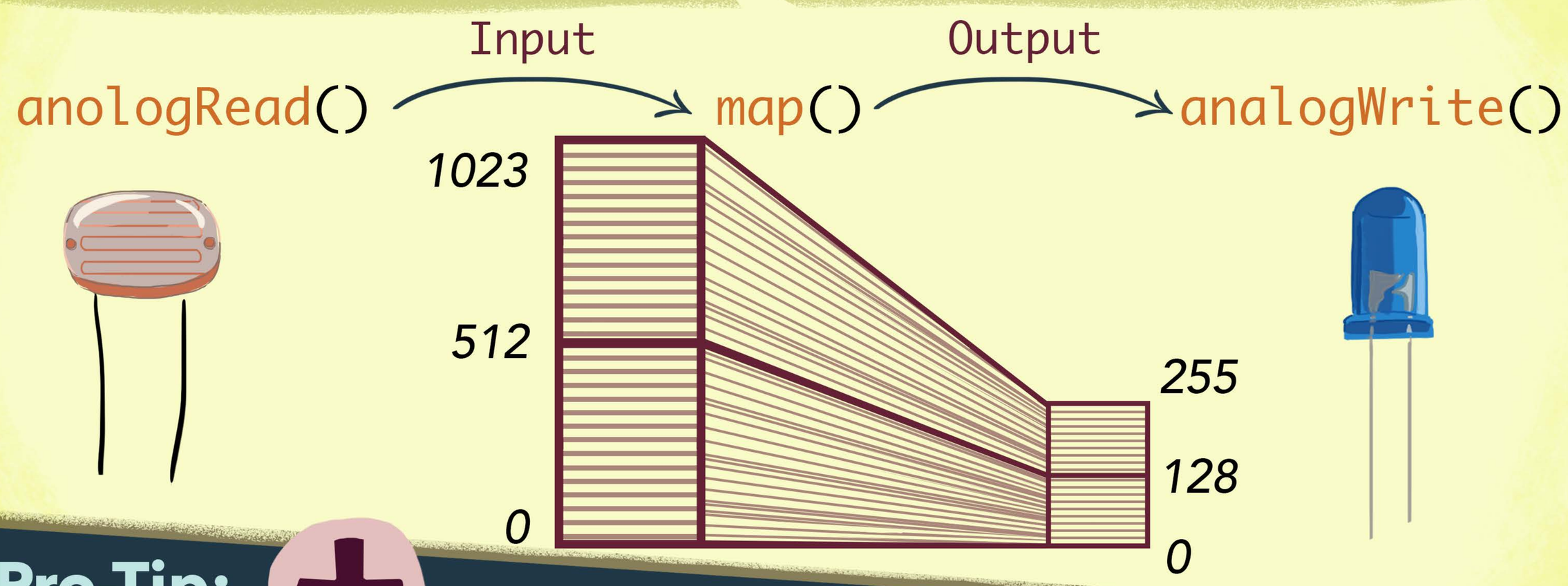
C4.4

```
int sensorValue = 0;
int sensorPin = 0;
int ledPin = 9;
int brightness = 0;

void setup(){
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop(){
  //Read the sensor and store it inside our
    "sensorValue" variable
  sensorValue = analogRead(sensorPin);

  //Use the map() function to change the values to a
    better range
  brightness = map(sensorValue, 0, 1023, 0, 255);

  //set the brightness of our LED
  analogWrite(ledPin, brightness);

  //Just in case we need to see what the sensor value is:
  Serial.println(sensorValue);
  delay(10);
}
```

More on this later



Let's take a closer look at the `map()` *function. We know our sensor values will range from 0-1023, but we can only set our LED to brightness values from 0-255. We need the* `map()` *function to help convert one range to the other.*



Pro-Tip:

The `map()` *function maps a range of integers to another range of integers. This means we can lose some precision when a large range gets mapped to a smaller range since no decimal places are allowed. For instance, in the case above, a value of 512 will map to 128 but so will 513 and 514. Therefore, leading to a loss in resolution.*

To use this map function we need to give it the right parameters.

The input range we want to map

The LOWEST value of the input range

The LOWEST value of the output range

```
map(sensorValue, 0, 1023, 0, 255);
```

The HIGHEST value of the input range

The HIGHEST value of the output range

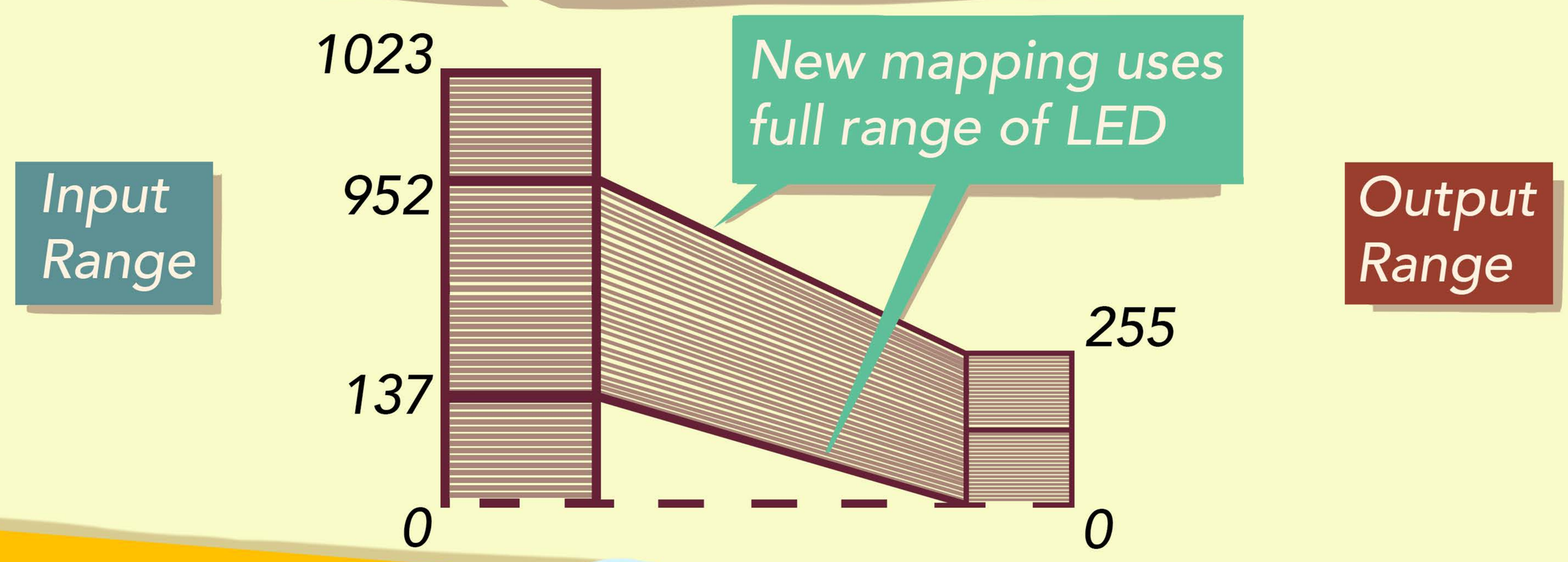
You'll find this works well, but we can still improve our nightlight. Since light in the room will rarely make the sensorValue get very close to 0 or 1023, let's use the ACTUAL range of values we are getting from the sensor.

Place the LOWEST value you measured here

```
brightness = map(sensorValue, 137 , 952 , 0, 255);
```

Place the HIGHEST value you measured here

This optimizes our night-light for the specific light conditions in the room. This calibration is especially important when you're in rooms that never get very dark or very bright.

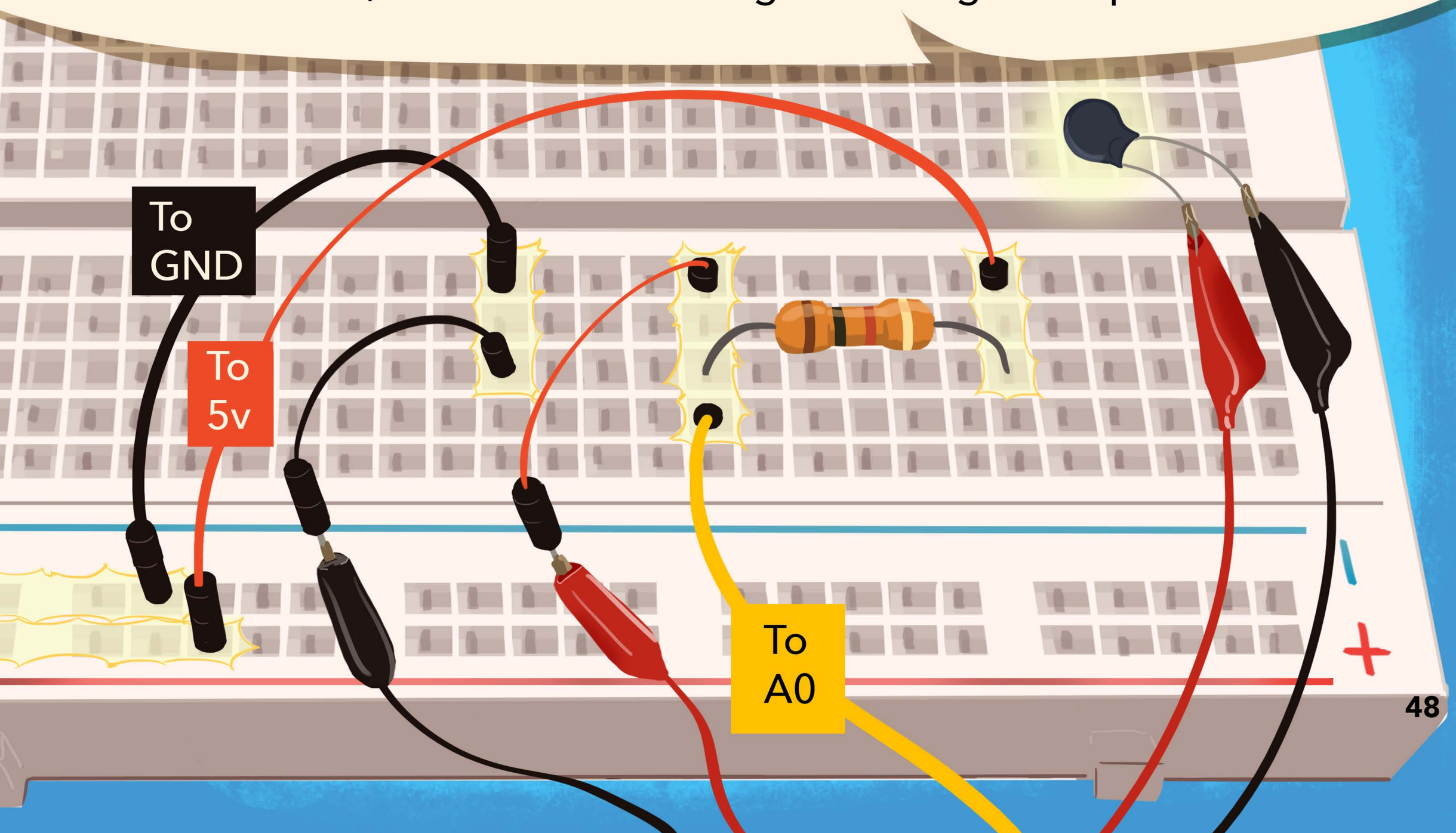


Challenge:

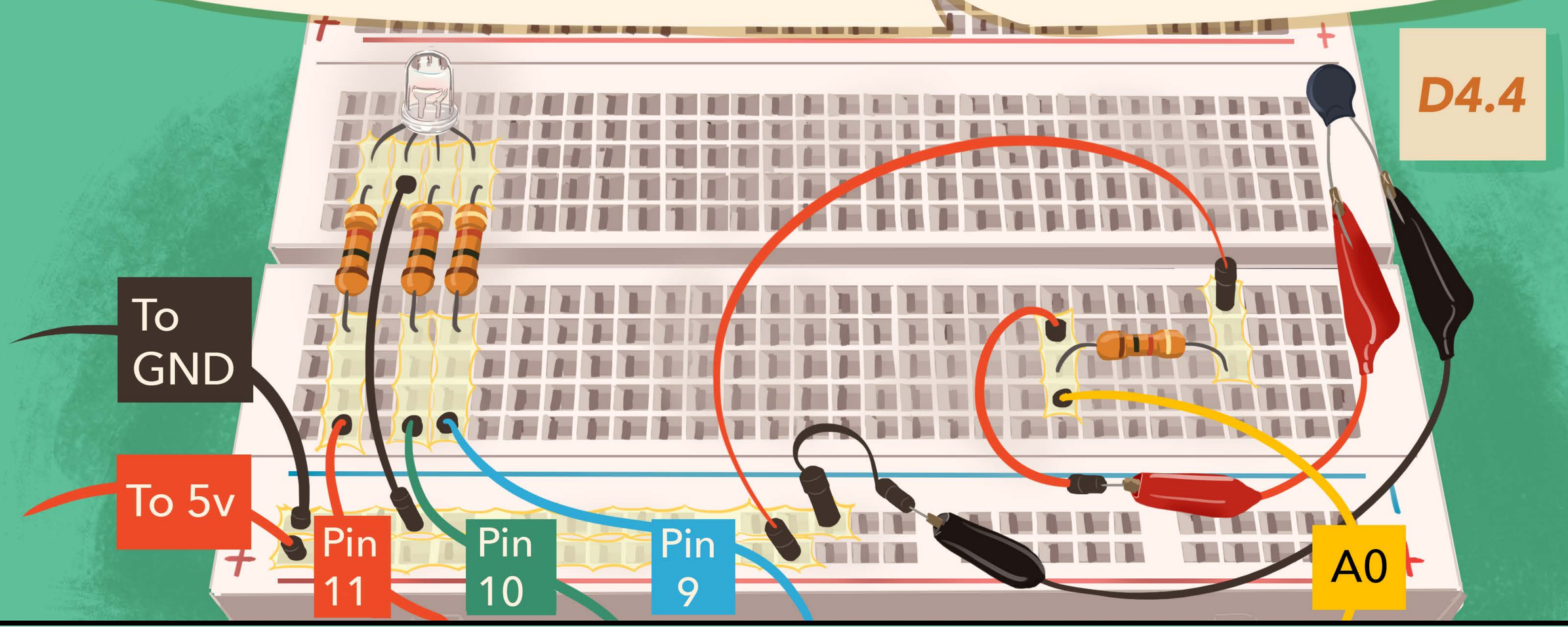
3c. Instead of a night-light, build a "day-light" circuit that is at full brightness when your room is at its brightest and off when your room is darkest

Note: There are two ways of doing this - one by only changing the night-light code and one by only changing the night-light circuit. Try both.

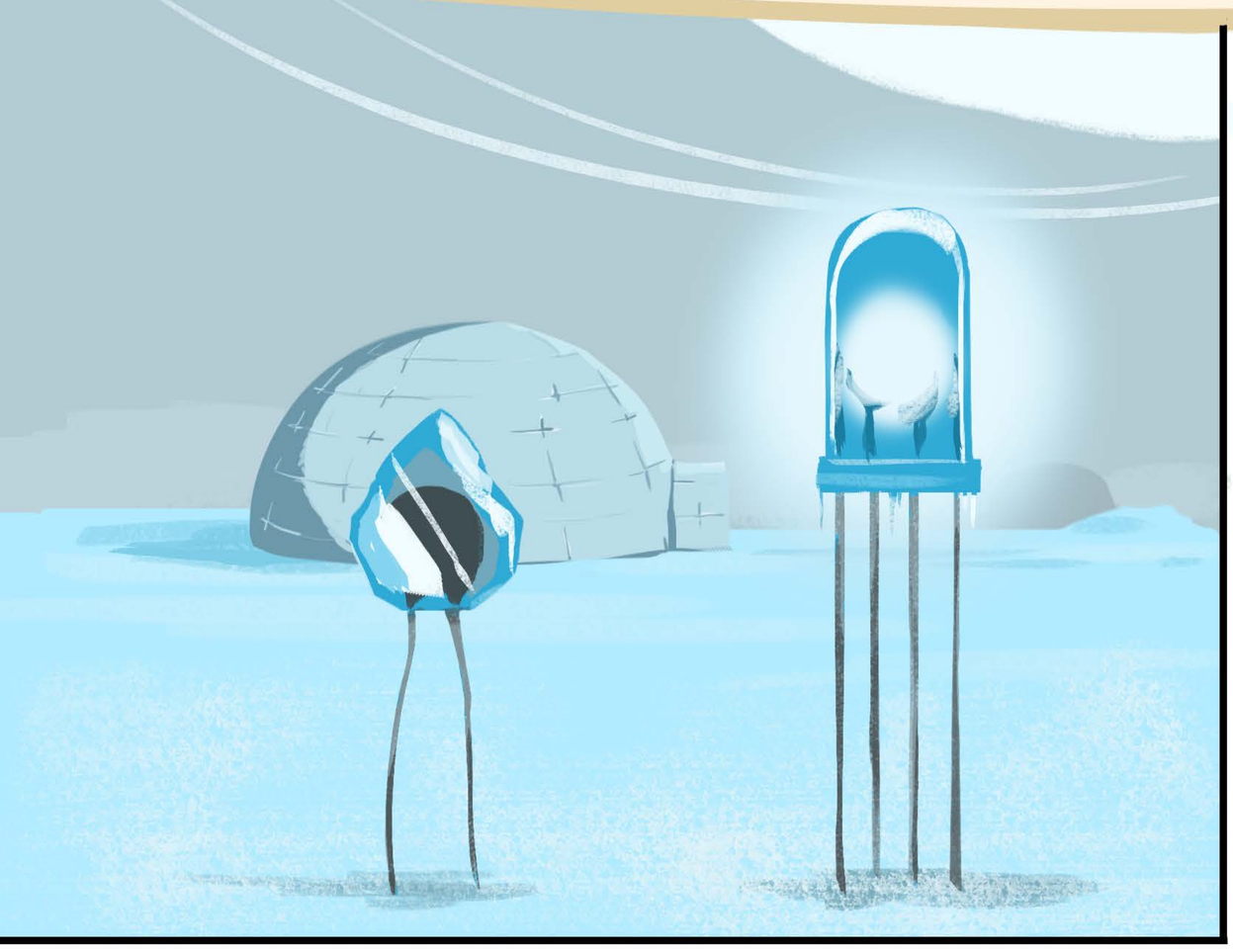
Using this voltage divider method for variable resistors comes in very handy. We can use it almost exactly as is for another sensor we have. Replace the photoresistor with your thermistor, i.e. temperature-sensitive resistor. Since we'll want our temperature sensor to be a bit more mobile, we'll connect it using some alligator clips.

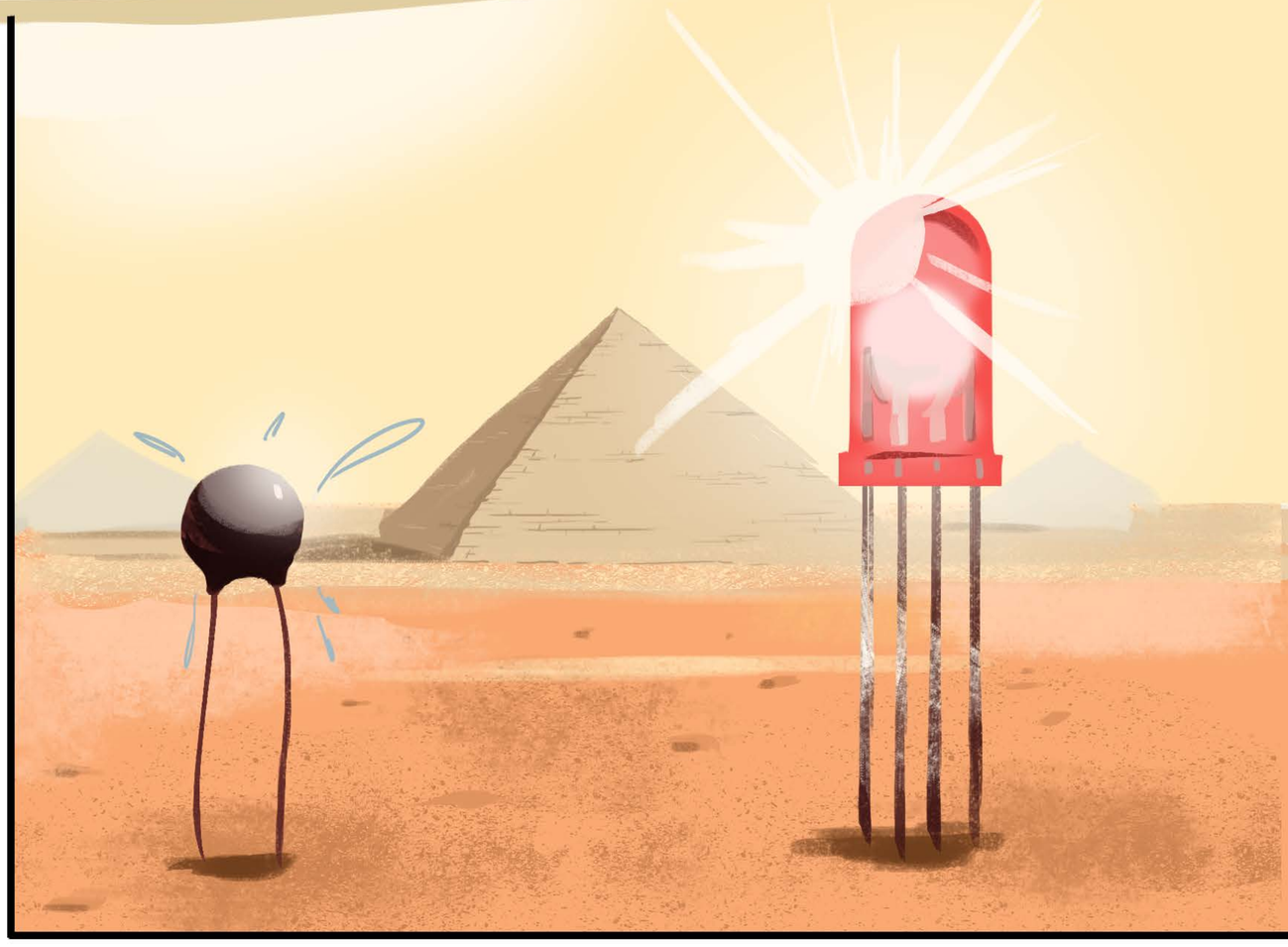


Also, let's use an RGB LED instead of a regular one so we can play around with some more colors.

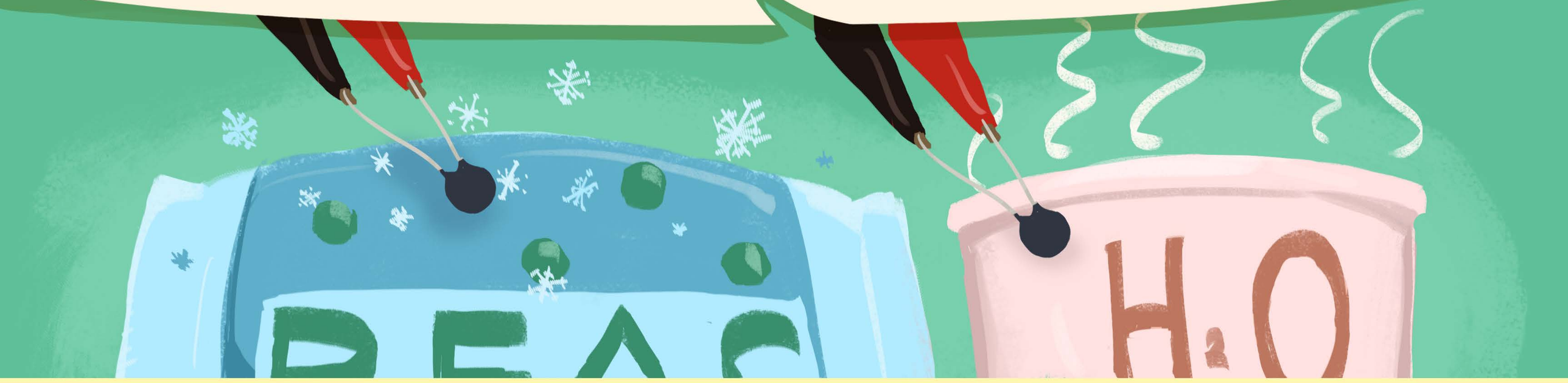


Let's use our thermistor to make a light that will change from blue to red as the temperature goes from cold to hot.





First, we'll need to know what values we're getting from the thermistor in hot and cold conditions. Try using something cold like a bag of frozen peas and hot like a hot cup of water to get extreme values and write them down.



 CAREFUL, DON'T SPILL ANYTHING ON YOUR CIRCUIT!

Upload this code to your UNO and change the values indicated to the ones you measured.

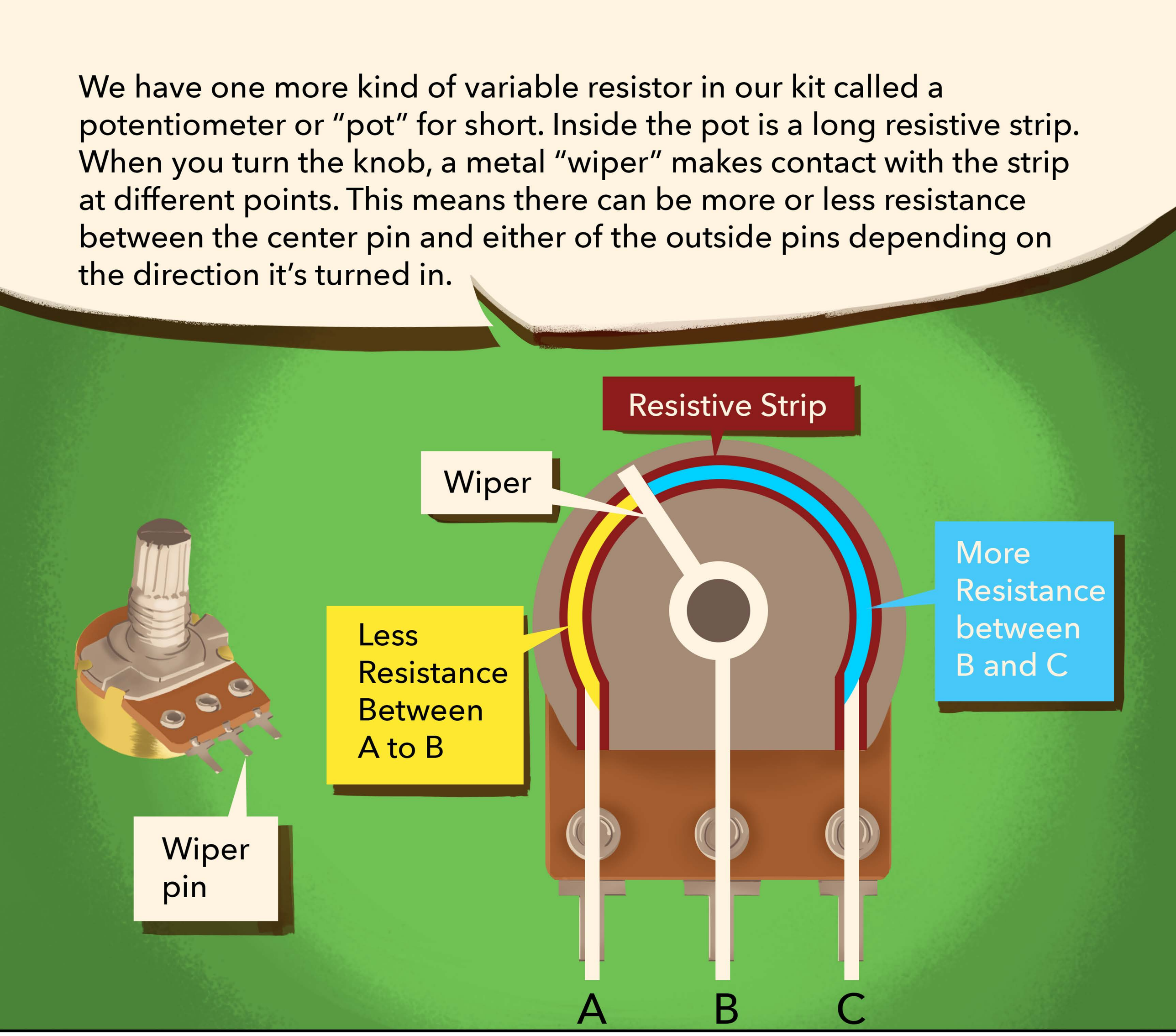
C4.3

```
int redBrightness = 0;
int blueBrightness = 0;
int redLedPin = 11;
int blueLedPin = 9;
int sensorValue = 0;
int sensorPin = 0;


//change these to your values
int coldValue = ______;
int hotValue =  ______;


void setup() {
  Serial.begin(9600);
  pinMode(redLedPin, OUTPUT);
  pinMode(blueLedPin, OUTPUT);

}

void loop() {
  //Read the sensor
  sensorValue = analogRead(sensorPin);


  //Change the sensor value to the right range for the RGB LED
redBrightness = map(sensorValue, coldValue, hotValue, 0, 255);
blueBrightness = map(sensorValue, coldValue, hotValue, 255, 0);


  //set the brightness of our RGB LED
  analogWrite(redLedPin, redBrightness);
  analogWrite(blueLedPin, blueBrightness);


  //Just in case we need to see what the sensor value is:
  Serial.println(sensorValue);
  delay(50);

}
```

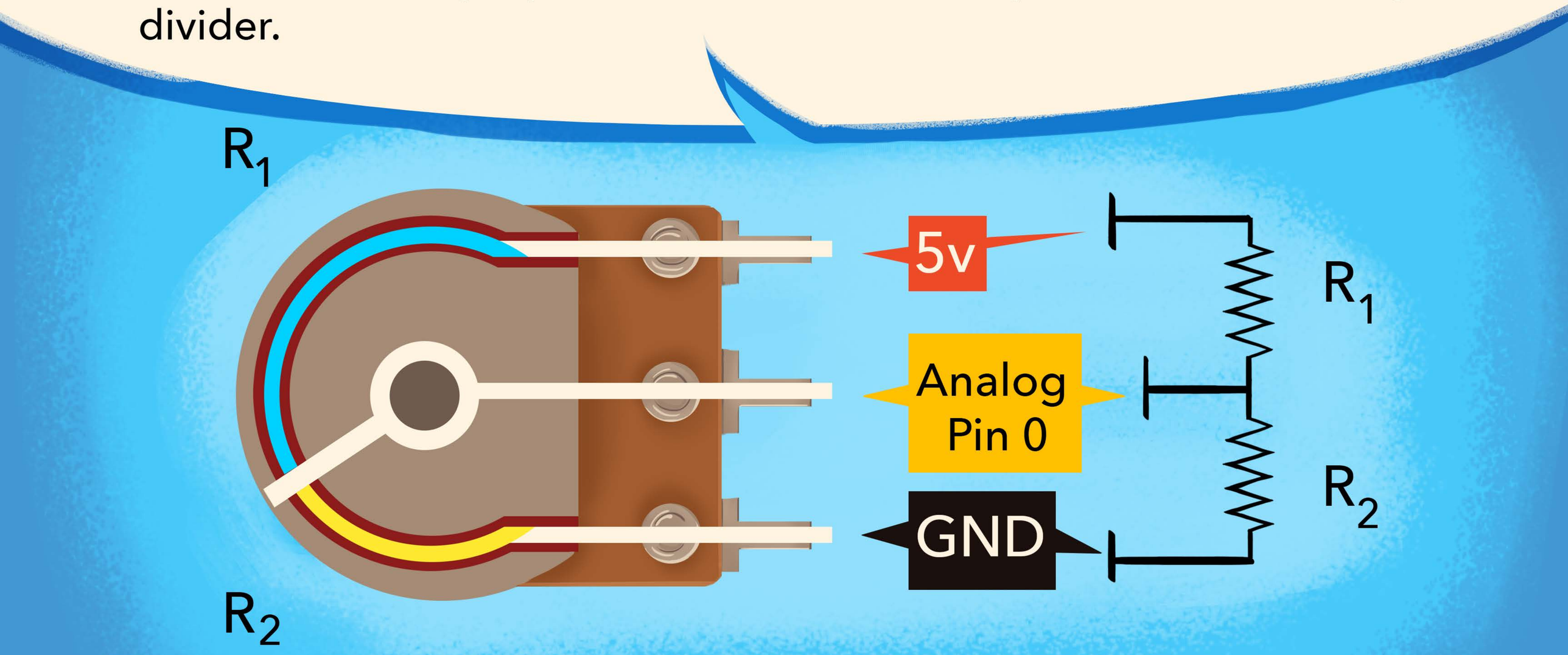
Your COLD value here

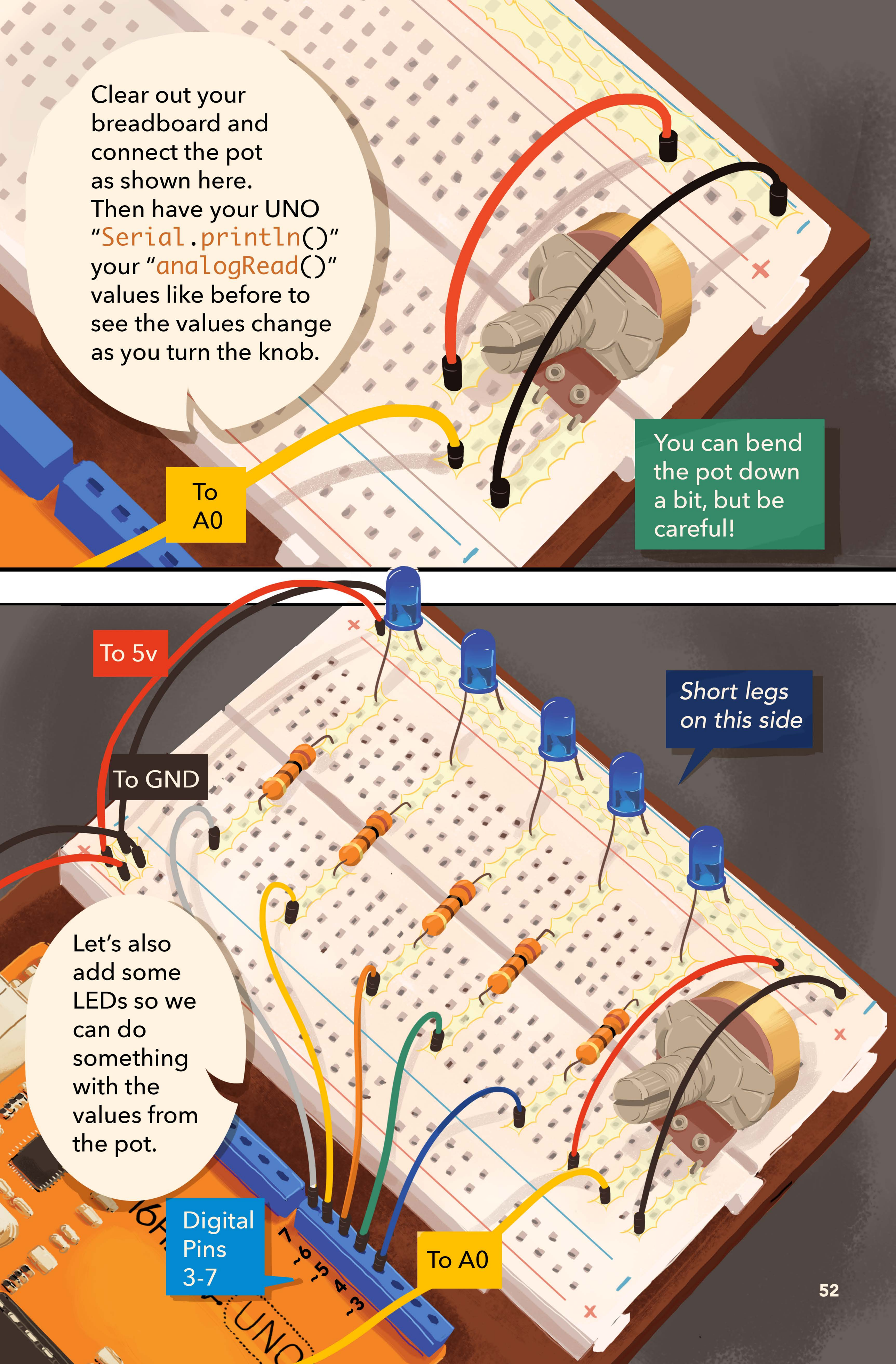
Your HOT value here

We have one more kind of variable resistor in our kit called a potentiometer or "pot" for short. Inside the pot is a long resistive strip. When you turn the knob, a metal "wiper" makes contact with the strip at different points. This means there can be more or less resistance between the center pin and either of the outside pins depending on the direction it's turned in.



The great thing about a pot is that it's like having 2 resistors built in, one between the left and the center pin and another between the center and the right pin. Therefore, it can easily be used as a voltage divider.




Clear out your breadboard and connect the pot as shown here. Then have your UNO "Serial.println()" your "analogRead()" values like before to see the values change as you turn the knob.
To A0
You can bend the pot down a bit, but be careful!
To 5v
To GND
Short legs on this side
Let's also add some LEDs so we can do something with the values from the pot.
Digital Pins 3-7
7
~6
~5
4
~3
To A0

```
int ledPin1 = 3;
int ledPin2 = 4;
int ledPin3 = 5;
int ledPin4 = 6;
int ledPin5 = 7;
int sensorValue = 0;
int sensorPin = 0;


void setup() {
  Serial.begin(9600);
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  pinMode(ledPin3, OUTPUT);
  pinMode(ledPin4, OUTPUT);
  pinMode(ledPin5, OUTPUT);
}

void loop(){
  //Read the sensor
  sensorValue = analogRead(sensorPin);

  //Turn on LEDs in succession as sensorValue increases
    and off if it decreases
if (sensorValue > 0 && sensorValue < 200) {
    digitalWrite(ledPin1, HIGH);    //turn ONE on
    digitalWrite(ledPin2, LOW);
    digitalWrite(ledPin3, LOW);
    digitalWrite(ledPin4, LOW);
    digitalWrite(ledPin5, LOW);


  } else if (sensorValue >= 200 && sensorValue < 400) {
    digitalWrite(ledPin1, HIGH);
    digitalWrite(ledPin2, HIGH);    //turn TWO on
    digitalWrite(ledPin3, LOW);
    digitalWrite(ledPin4, LOW);
    digitalWrite(ledPin5, LOW);

  } else if (sensorValue >= 400 && sensorValue < 600) {
    digitalWrite(ledPin1, HIGH);
    digitalWrite(ledPin2, HIGH);
    digitalWrite(ledPin3, HIGH);    //turn THREE on
    digitalWrite(ledPin4, LOW);
    digitalWrite(ledPin5, LOW);
```

Upload this code to your UNO and see how we use multiple if statements to divide the pot into 5 different sections, each doing different things to the LEDs.

C4.4

The ">=" means "greater than or equal". If we only used ">", the UNO wouldn't know what to do if sensorValue was exactly 200.

Continued on next page.

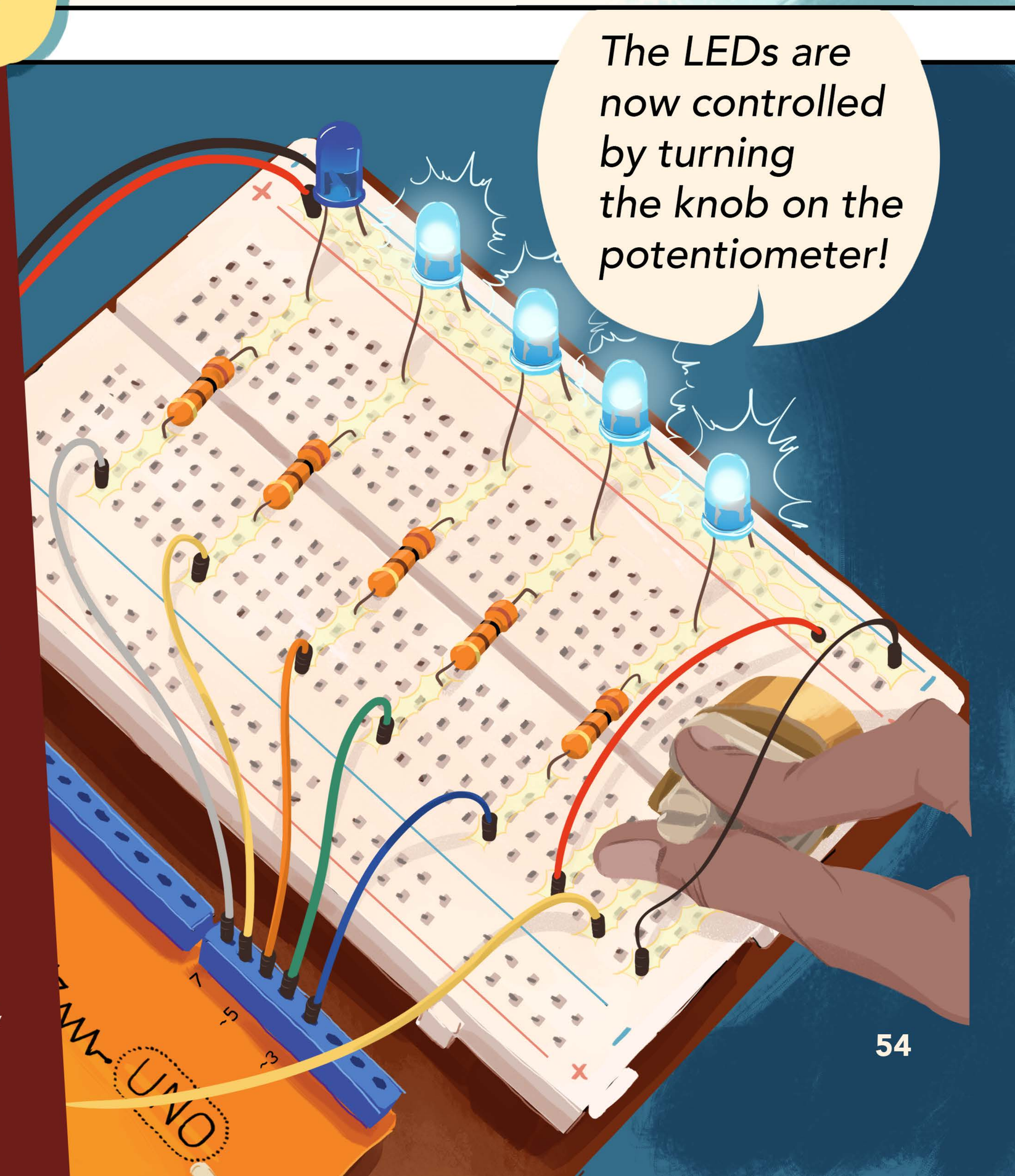
Continued from last page.


```
  } else if (sensorValue >= 600 && sensorValue < 800) {
    digitalWrite(ledPin1, HIGH);
    digitalWrite(ledPin2, HIGH);
    digitalWrite(ledPin3, HIGH);
    digitalWrite(ledPin4, HIGH);    //turn FOUR on
    digitalWrite(ledPin5, LOW);

  } else if (sensorValue >= 800) {
    digitalWrite(ledPin1, HIGH);
    digitalWrite(ledPin2, HIGH);
    digitalWrite(ledPin3, HIGH);
    digitalWrite(ledPin4, HIGH);
    digitalWrite(ledPin5, HIGH);    //turn FIVE on
  }


  //Just in case we need to see what the sensor value is:
  Serial.println(sensorValue);
  delay(50);
}
```

No need to have "&& sensorValue < 1023" because anything greater than 800 is ok for the last section of the pot.

Helpful Hint:

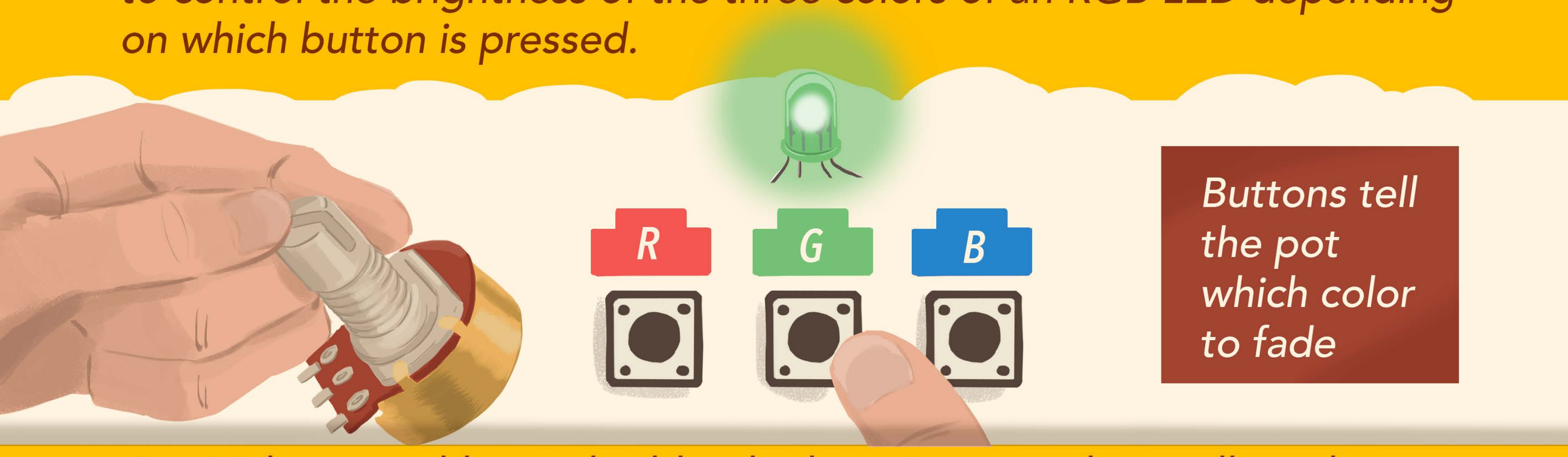
*Else statements allow us to execute a specific set of commands if the condition in the preceding `if()` statement is **NOT** satisfied. Because of this, the UNO won't even read anything in the else statement if the previous condition has been satisfied. The code would still work if we removed all the else statements, but it would make the UNO check all the other conditions unnecessarily, making it less efficient.*



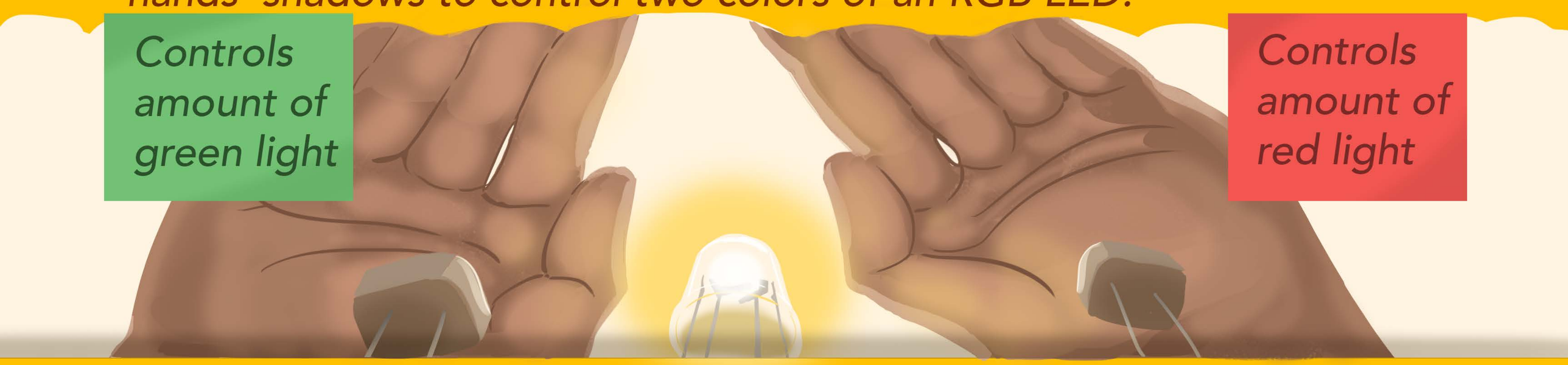
All of the things we've done in this tutorial have simply taken an input, made some decisions or changes to a number and then controlled an output. This is the core of almost every? electronic device in the world!

Challenge:

4d. *With 3 buttons and a potentiometer, make a circuit that allows you to control the brightness of the three colors of an RGB LED depending on which button is pressed.*



4e. *Make a touchless color blender by using two photocells and your hands' shadows to control two colors of an RGB LED.*



4f. *Use a thermistor and an RGB LED to make a device that will change colors when your hot chocolate is not too hot or too cold to drink.*

REFERENCE

The following pages have been assembled and illustrated to provide you with a clear, unambiguous resource for assembling the circuits refered to previously in this guide. Each section is denoted with a label to let you know where to find this activity in the tutorials. ***D3.2***, for example, is the **D**iagram for tutorial **3**, activity **2**.

For each of these activities, you will find a ***parts list***, an easy to read breadboard ***wiring diagram*** and ***schematic diagram*** to help you connect your circuits to the universally-used language and conventions of electrical engineers. You can find the info you need for each activity by looking for

You will also find a link to an online download of all the Arduino code you will need to follow along with the tutorials. Assembled for your convenience, the code is organized using the same notation as the diagram pages.

Good luck!

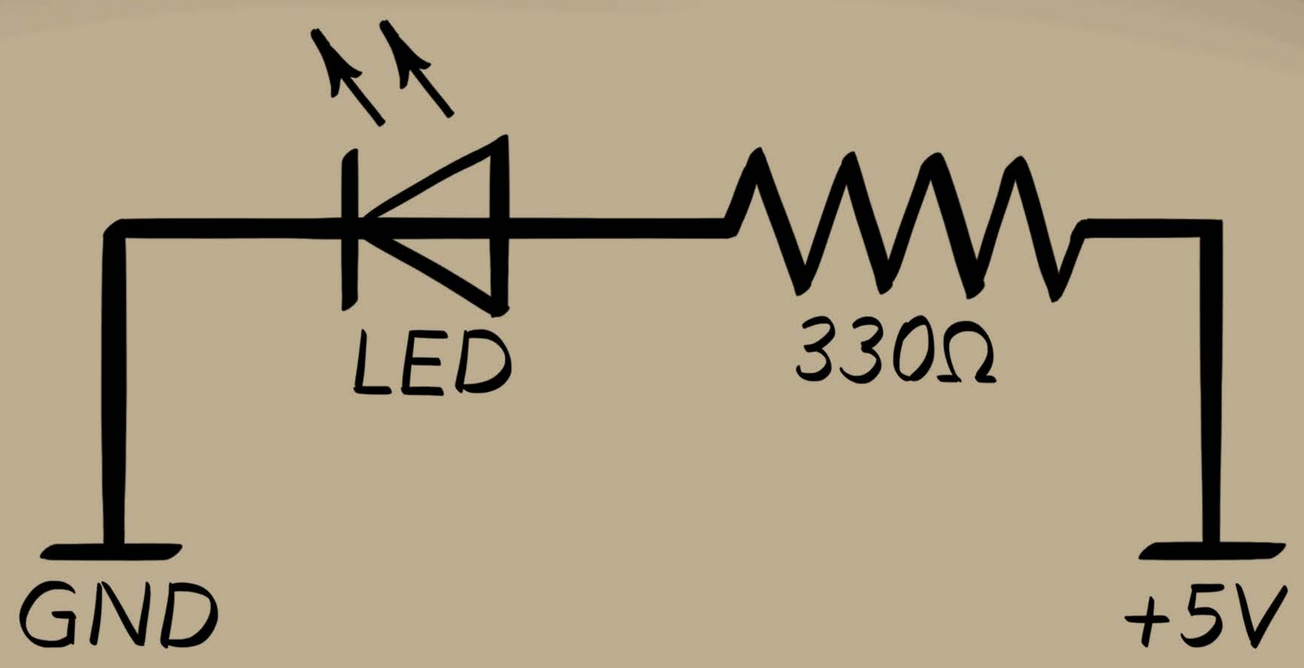
ARDUINO CODE

To download the sample code for the tutorials in this guide, be sure to visit:

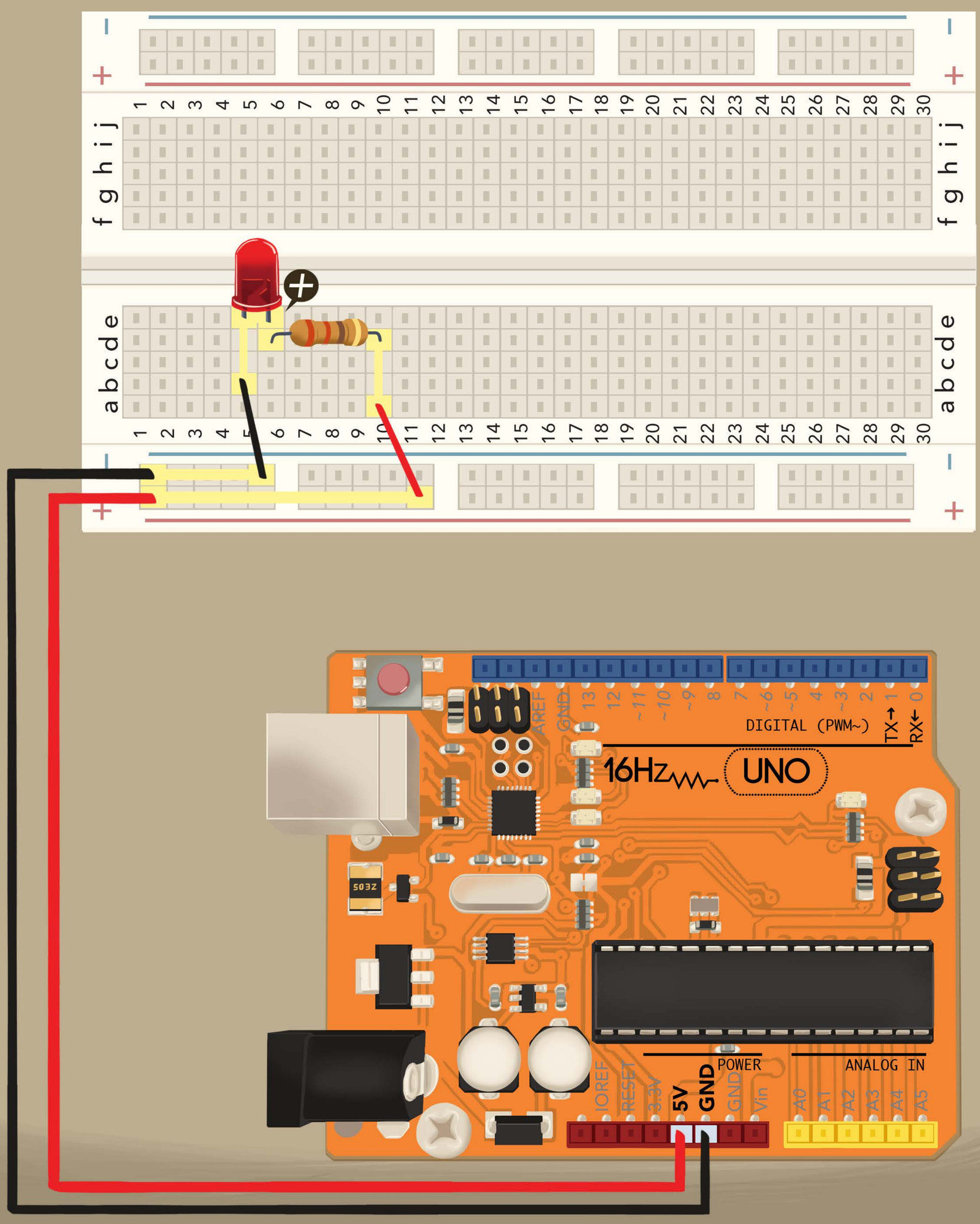
www.bit.ly/16hzcode

Schematic

D0.1

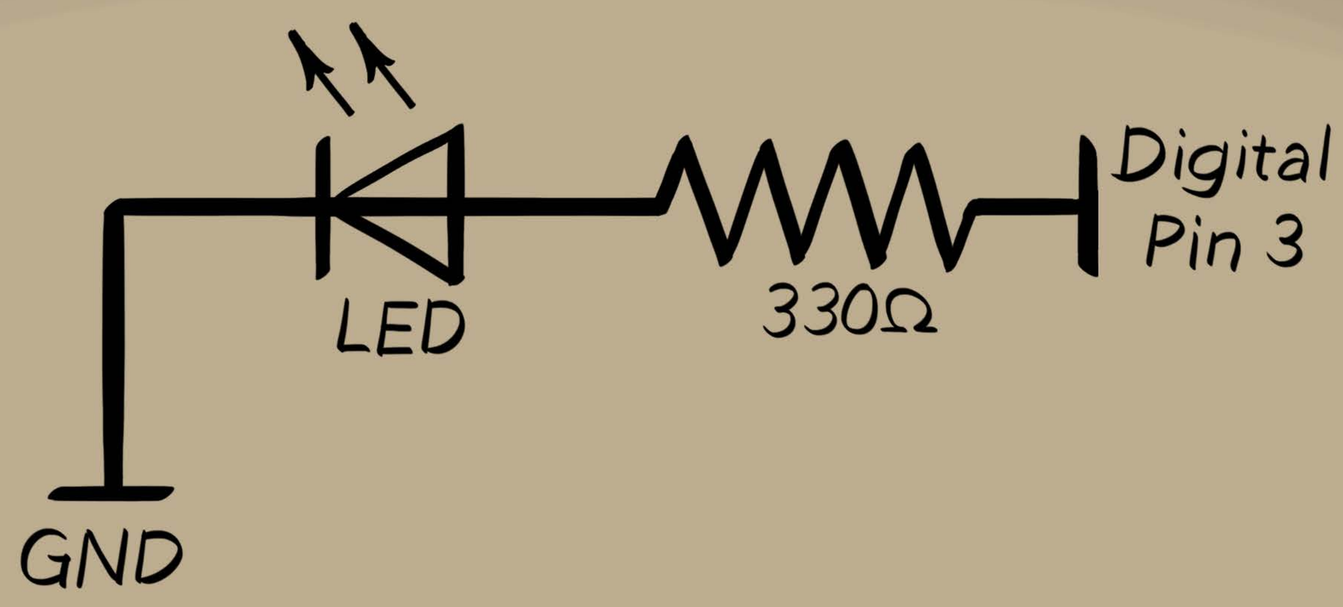


Wiring Diagram

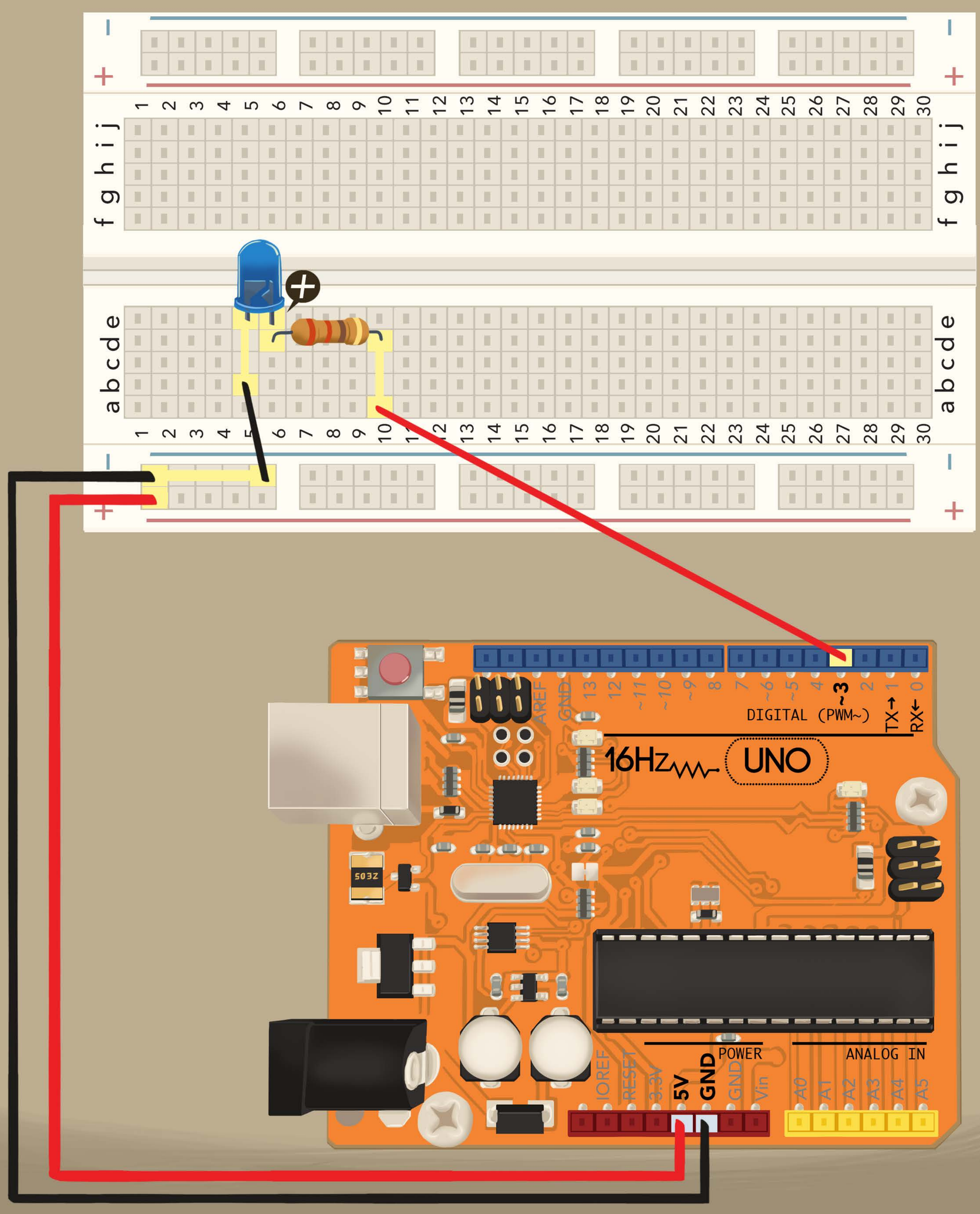


Schematic

D1.1

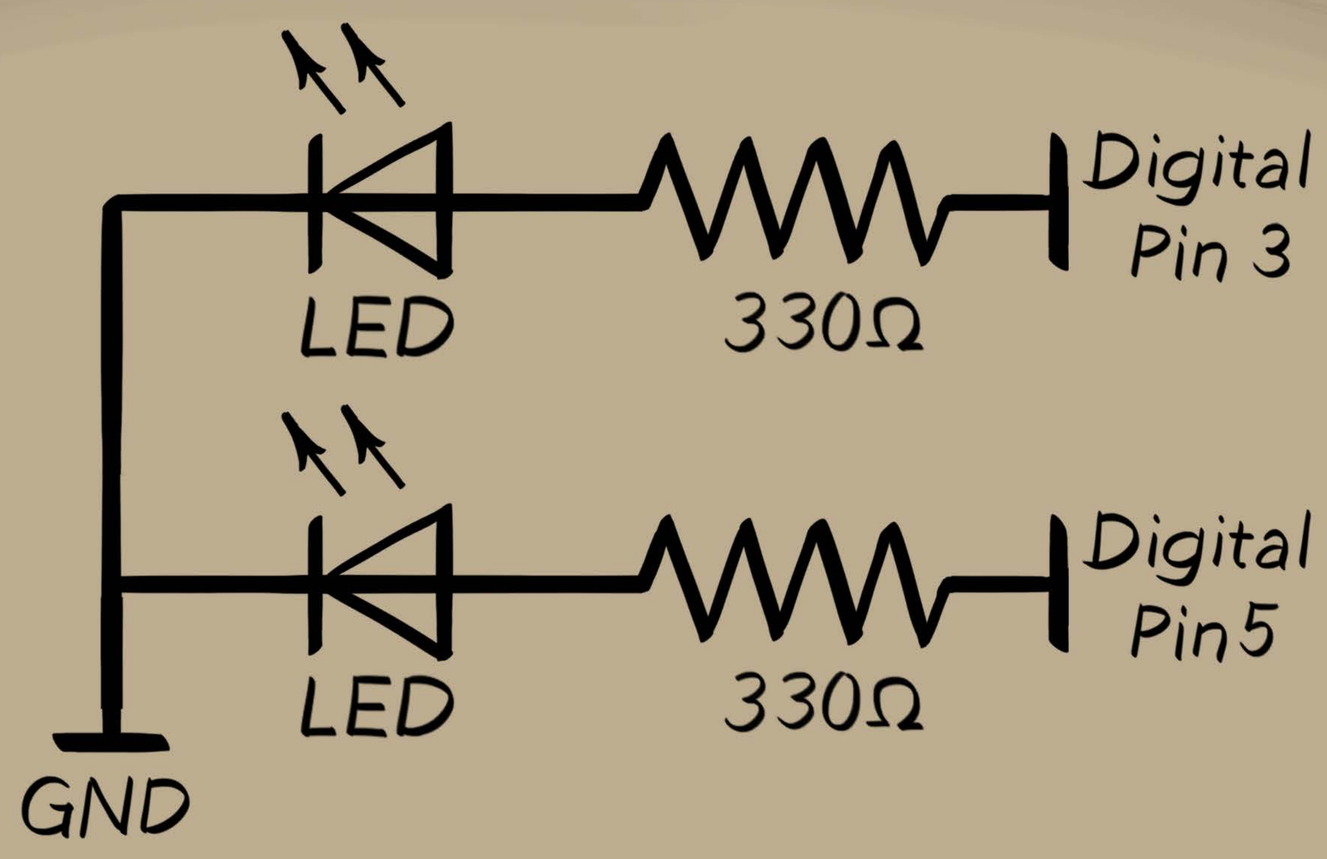


Wiring Diagram

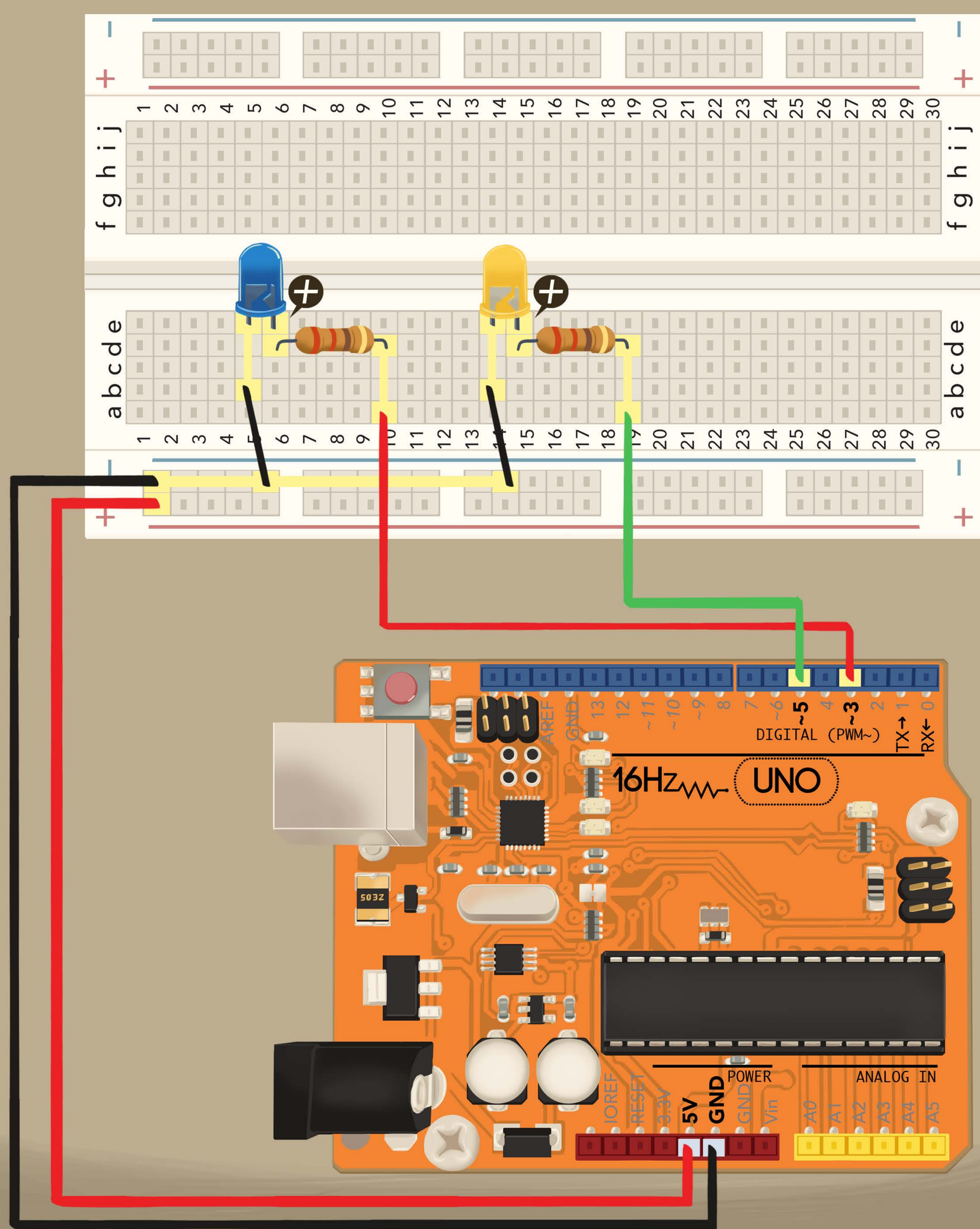


Schematic

D1.2

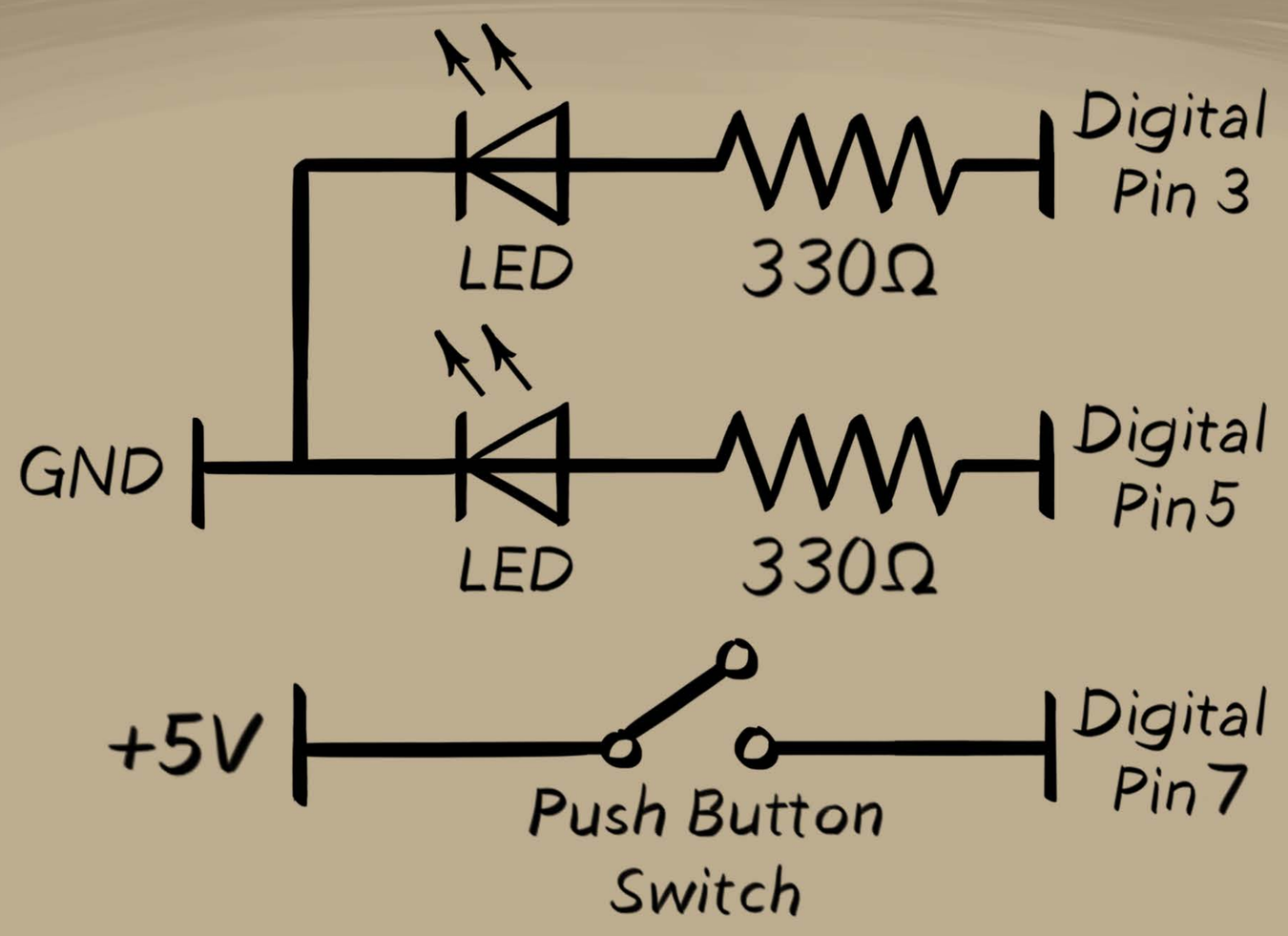


Wiring Diagram

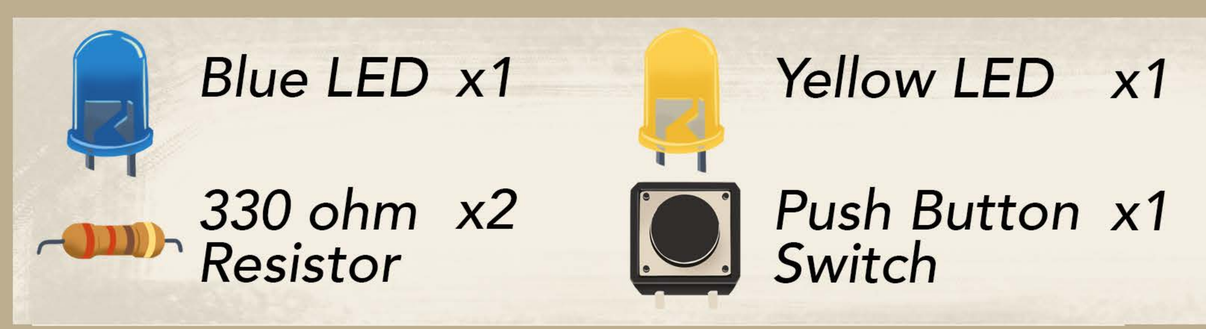


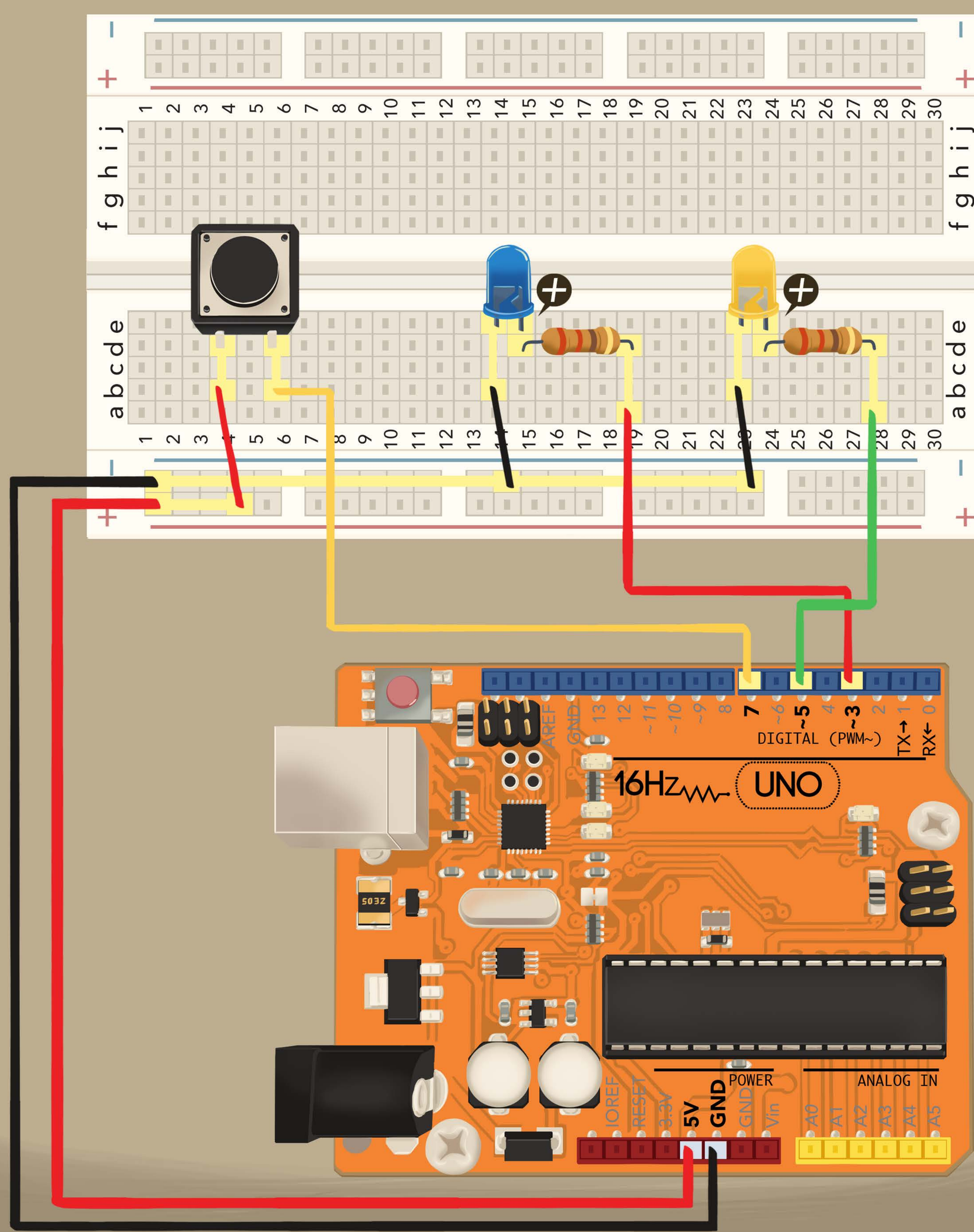
Schematic

D2.1



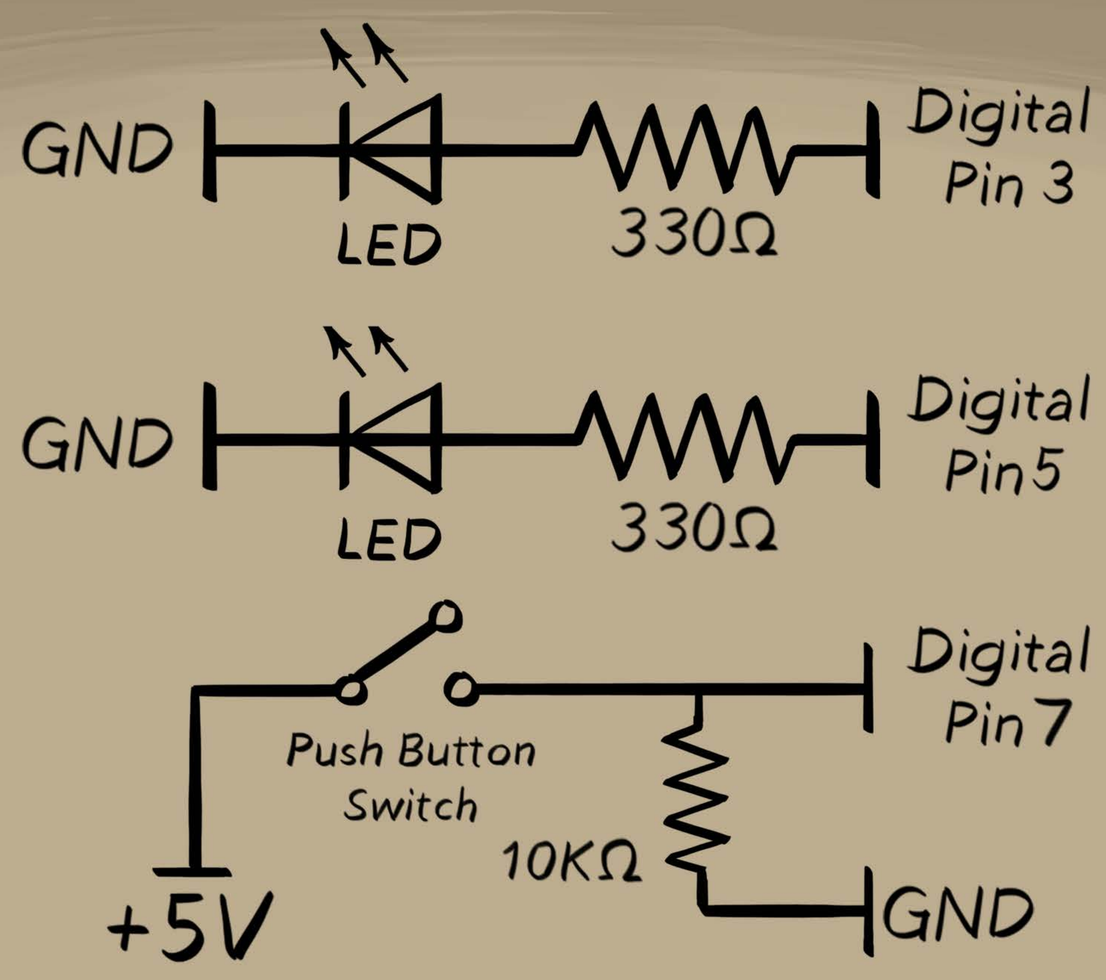
Wiring Diagram



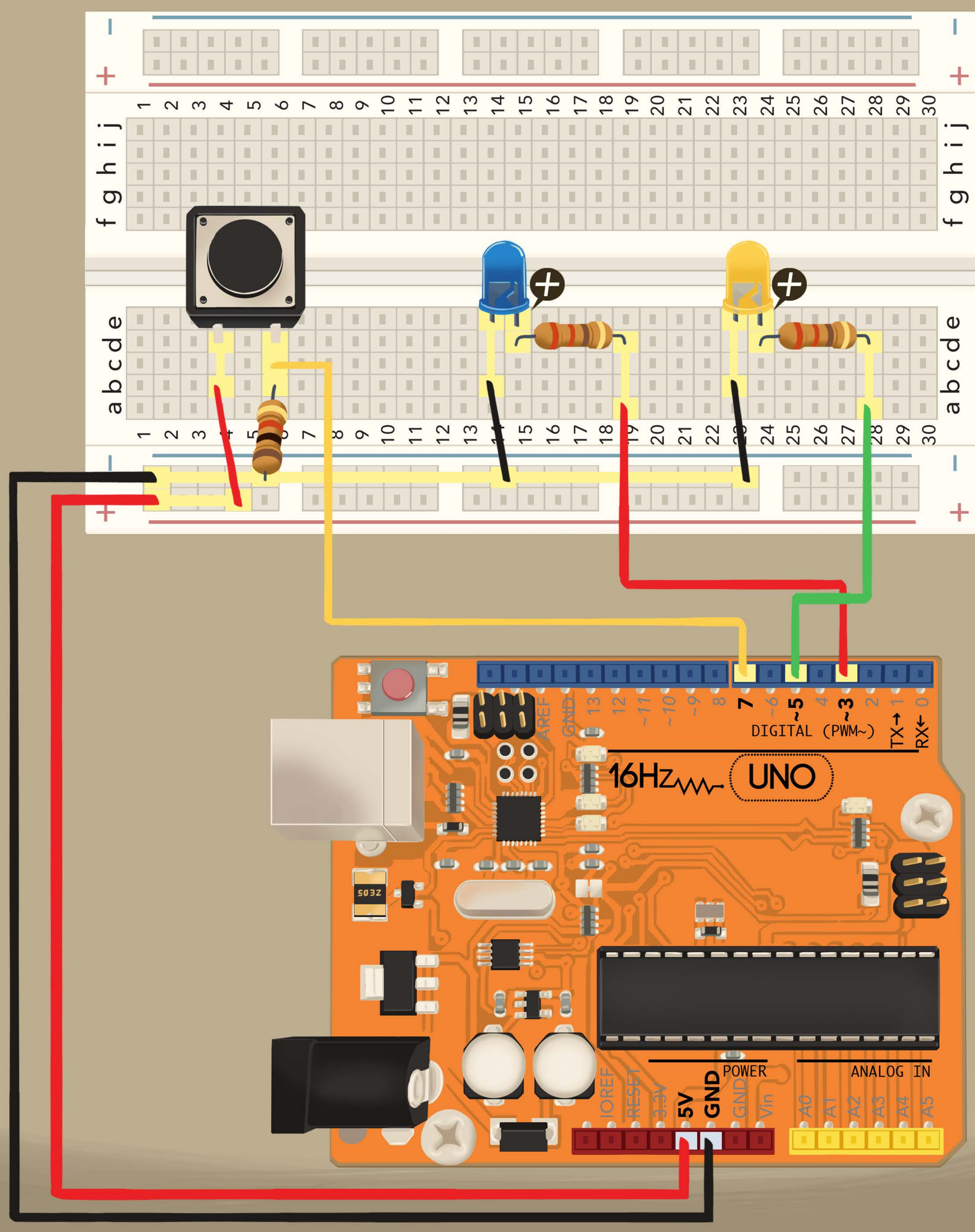


Schematic

D2.2

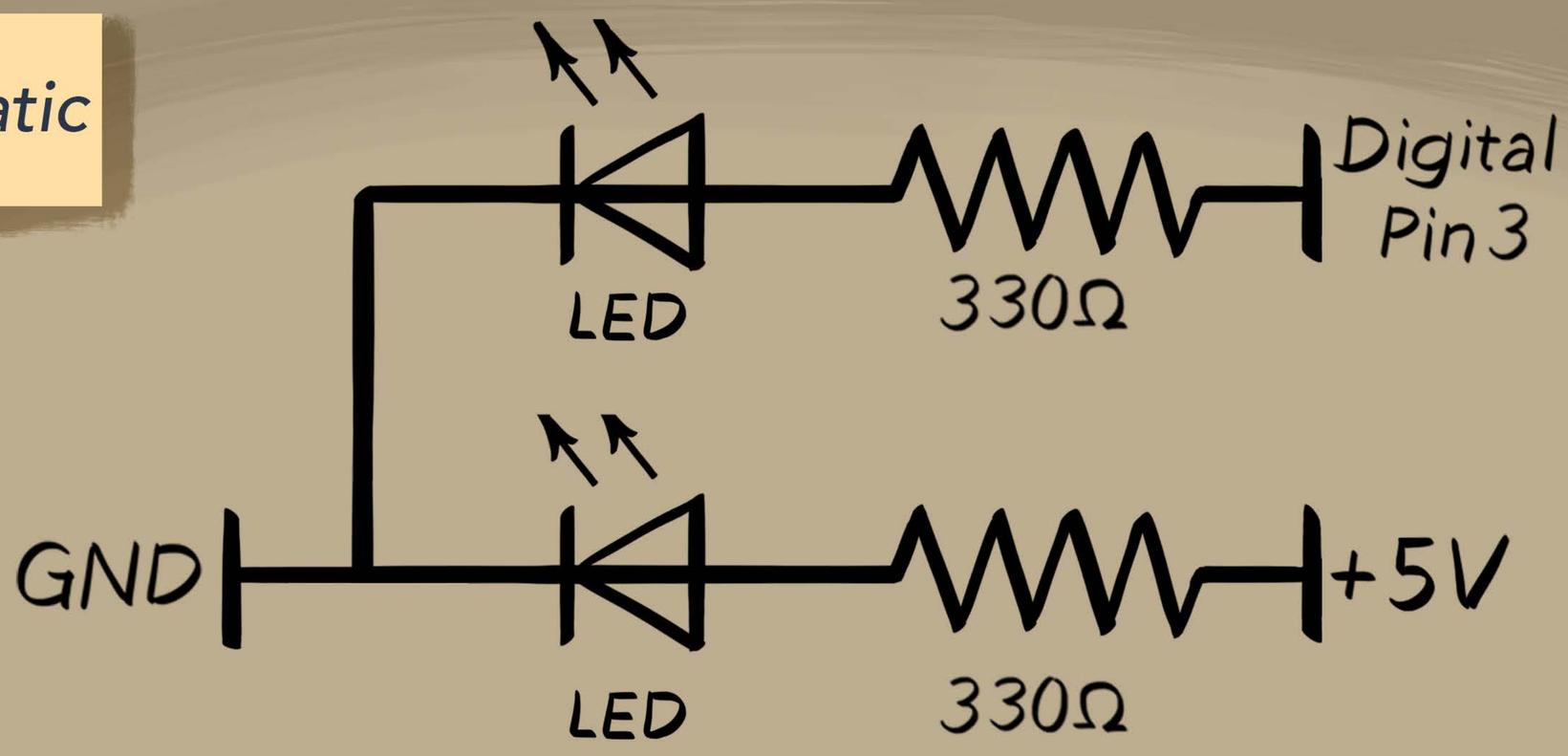


Wiring Diagram

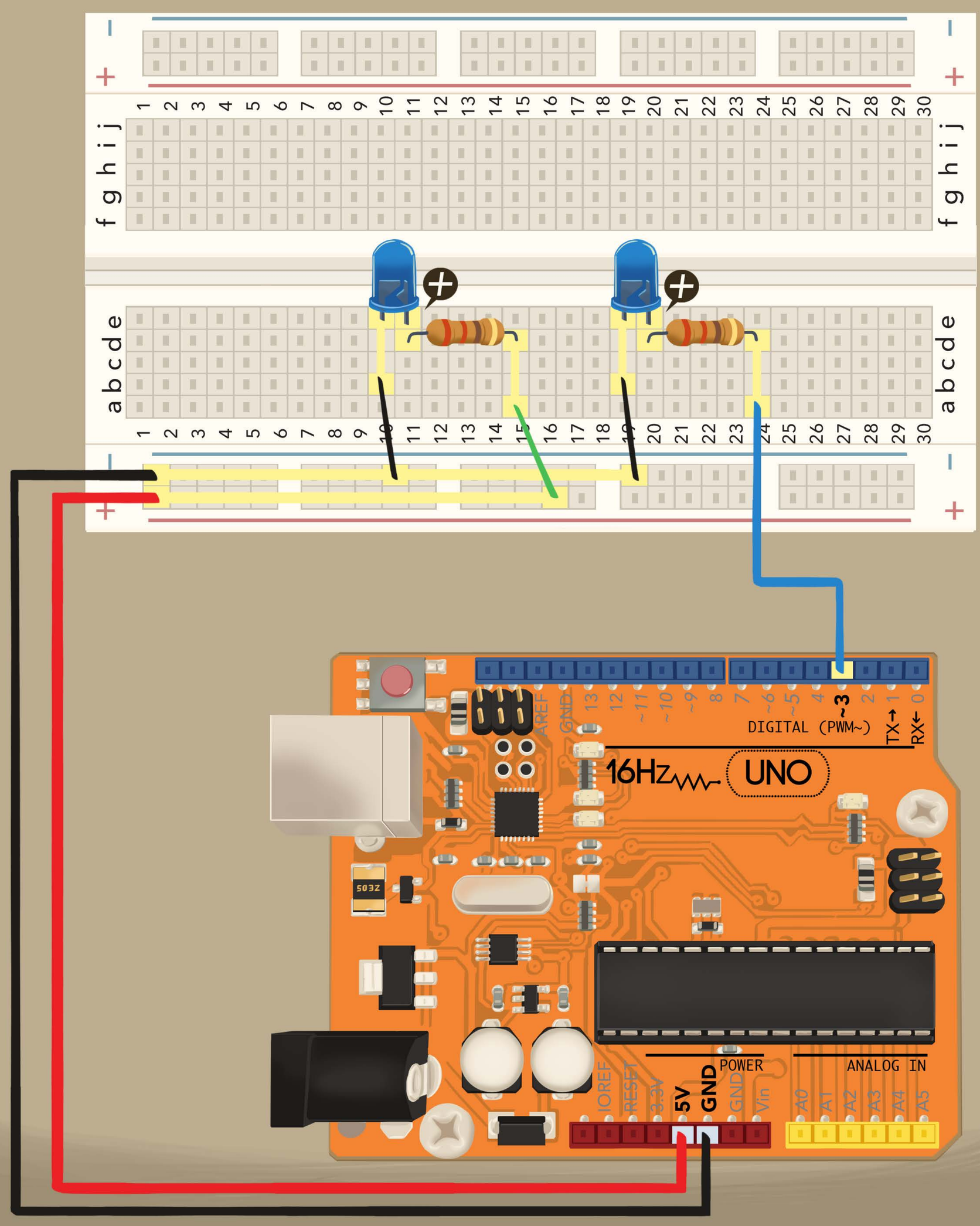


Schematic

D3.1

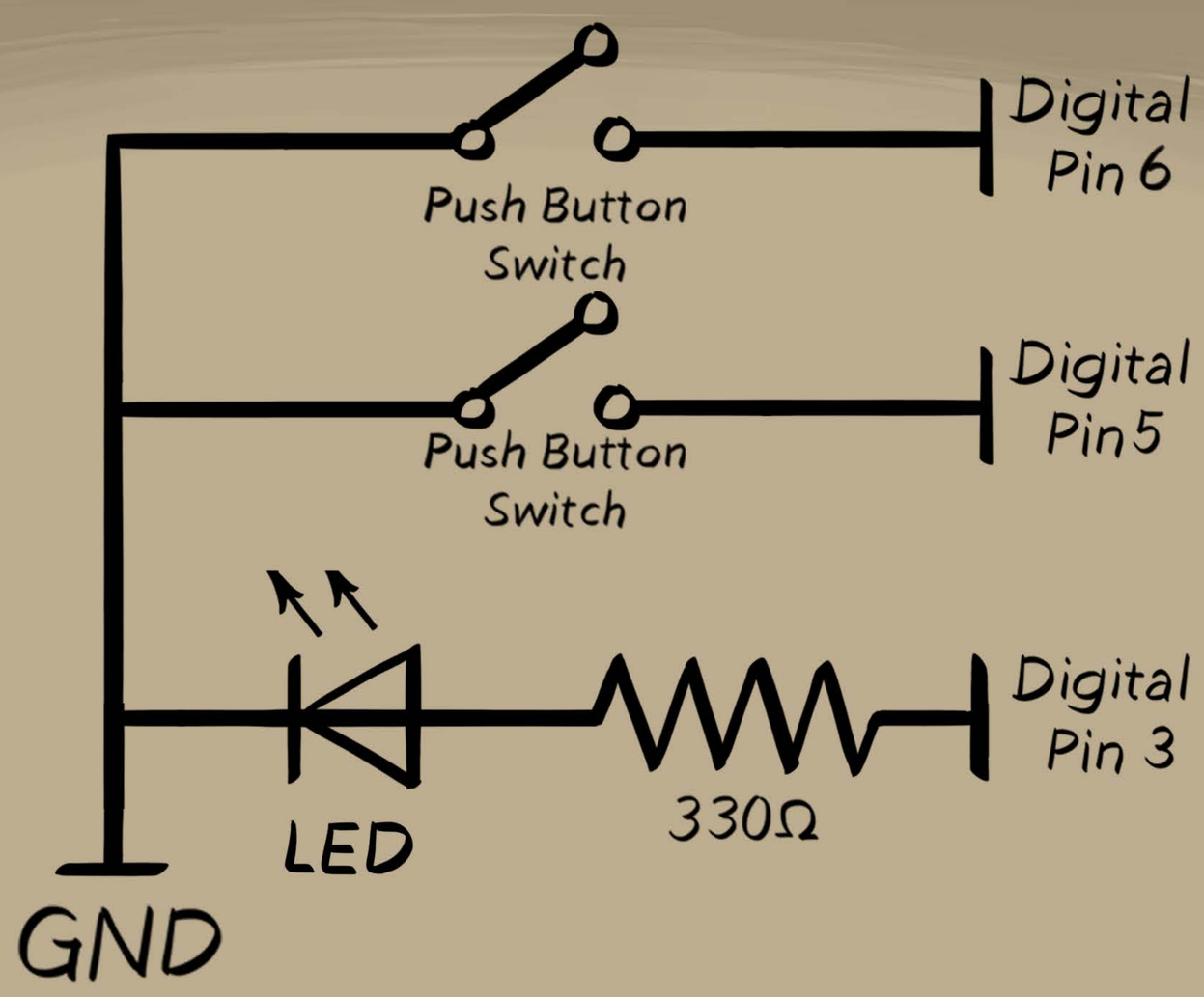


Wiring Diagram

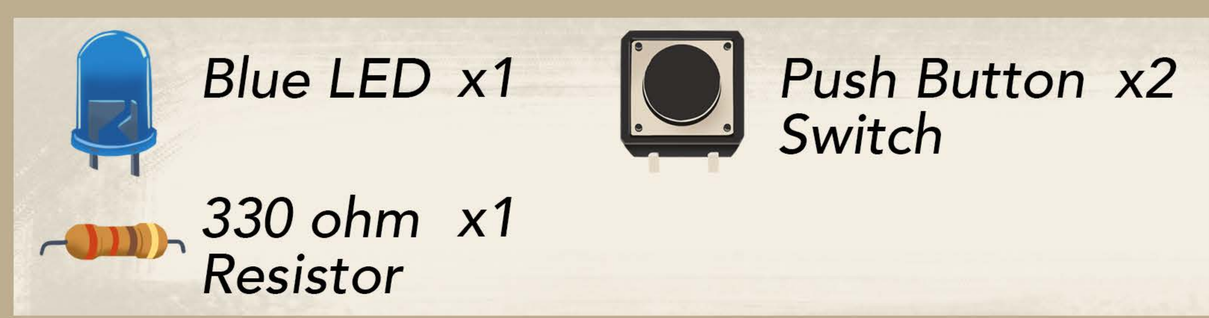


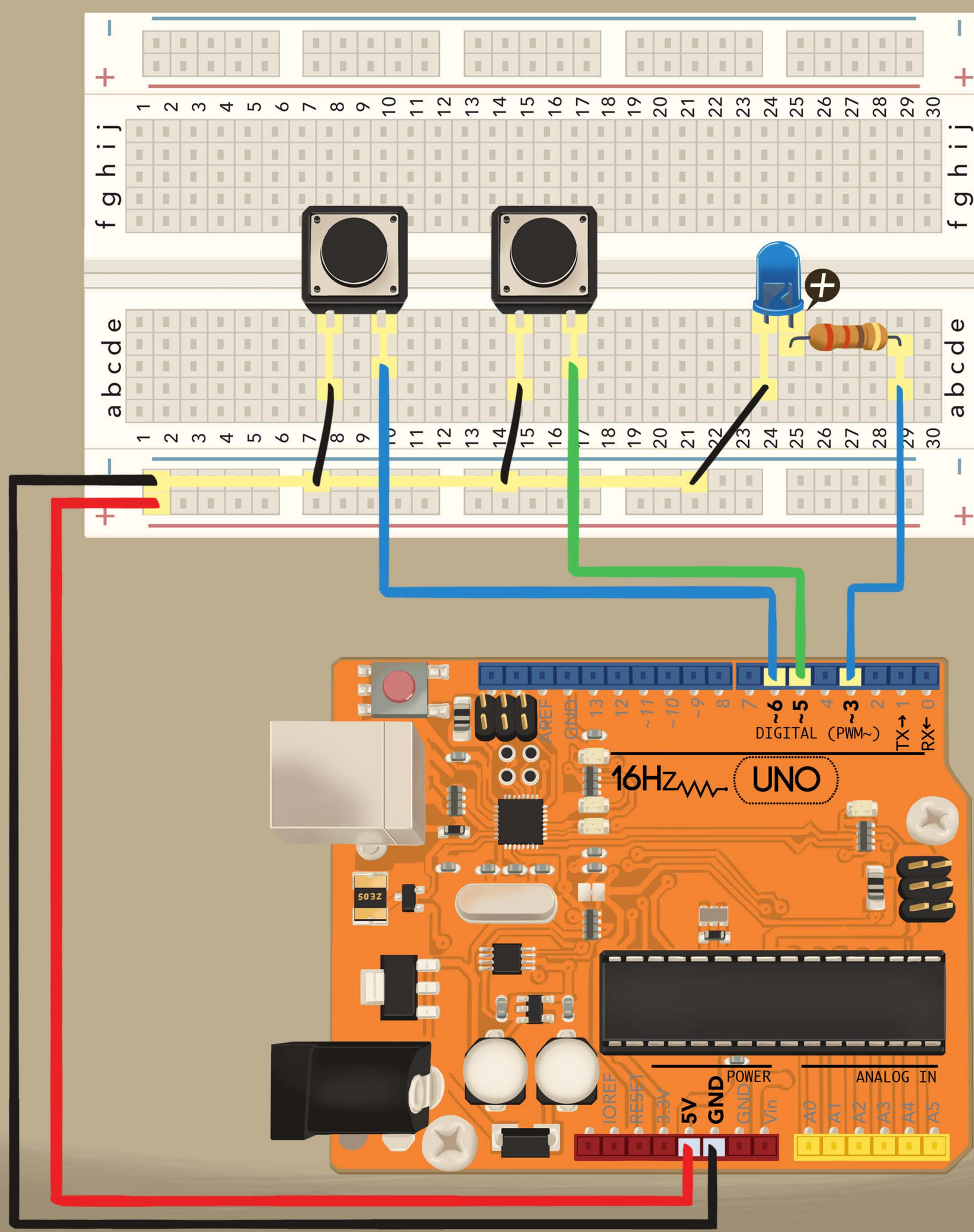
Schematic

D3.2



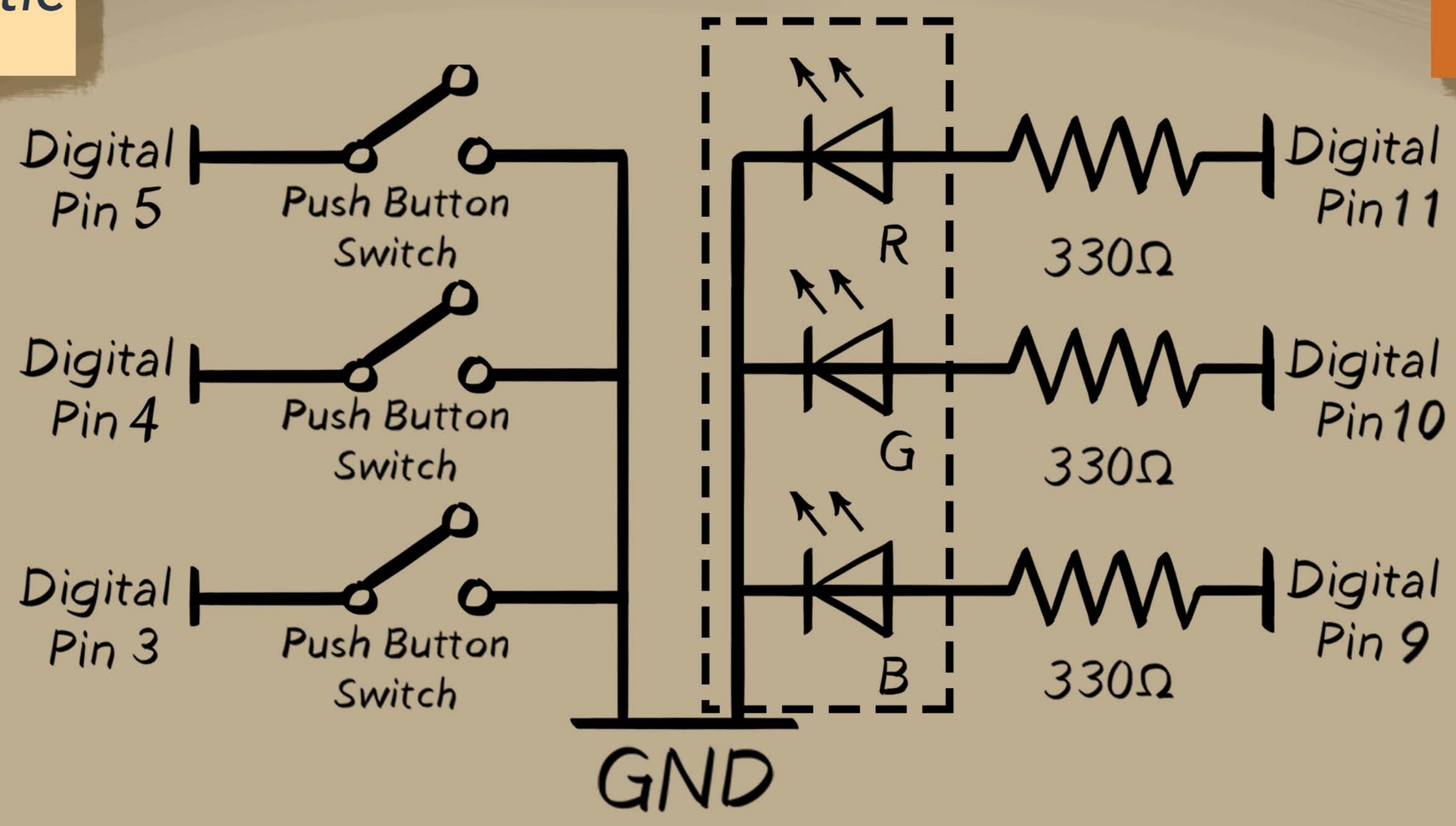
Wiring Diagram



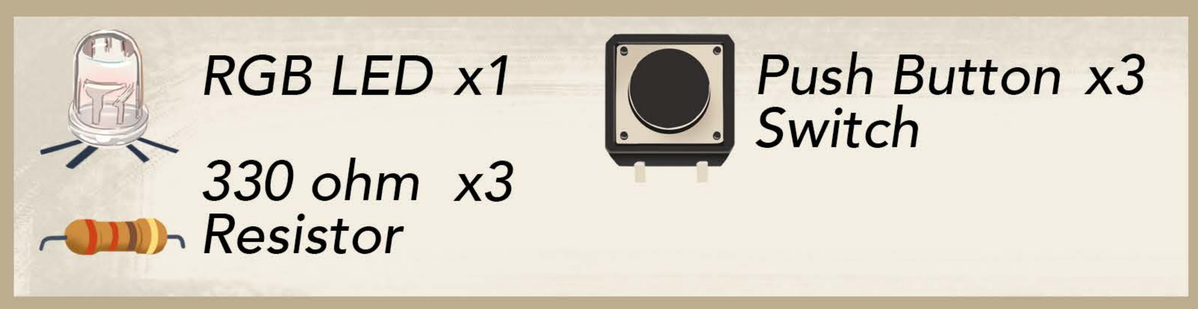


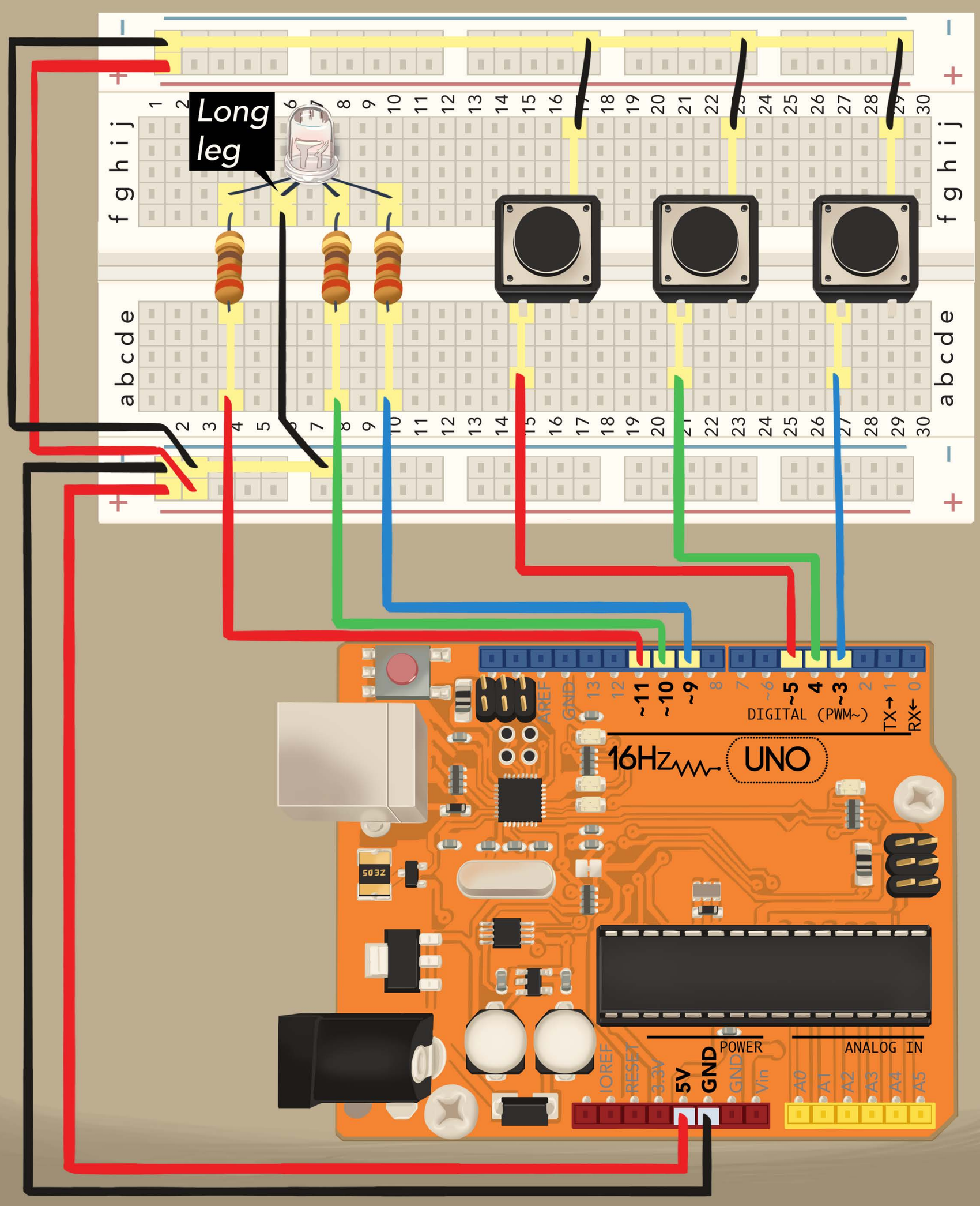
Schematic

D3.3



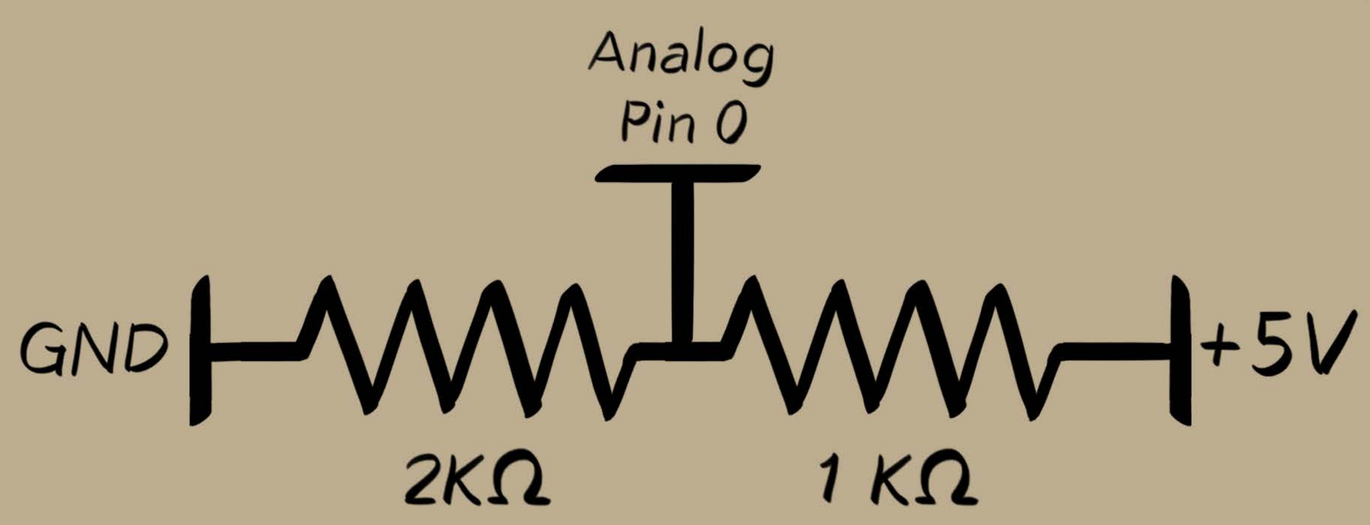
Wiring Diagram



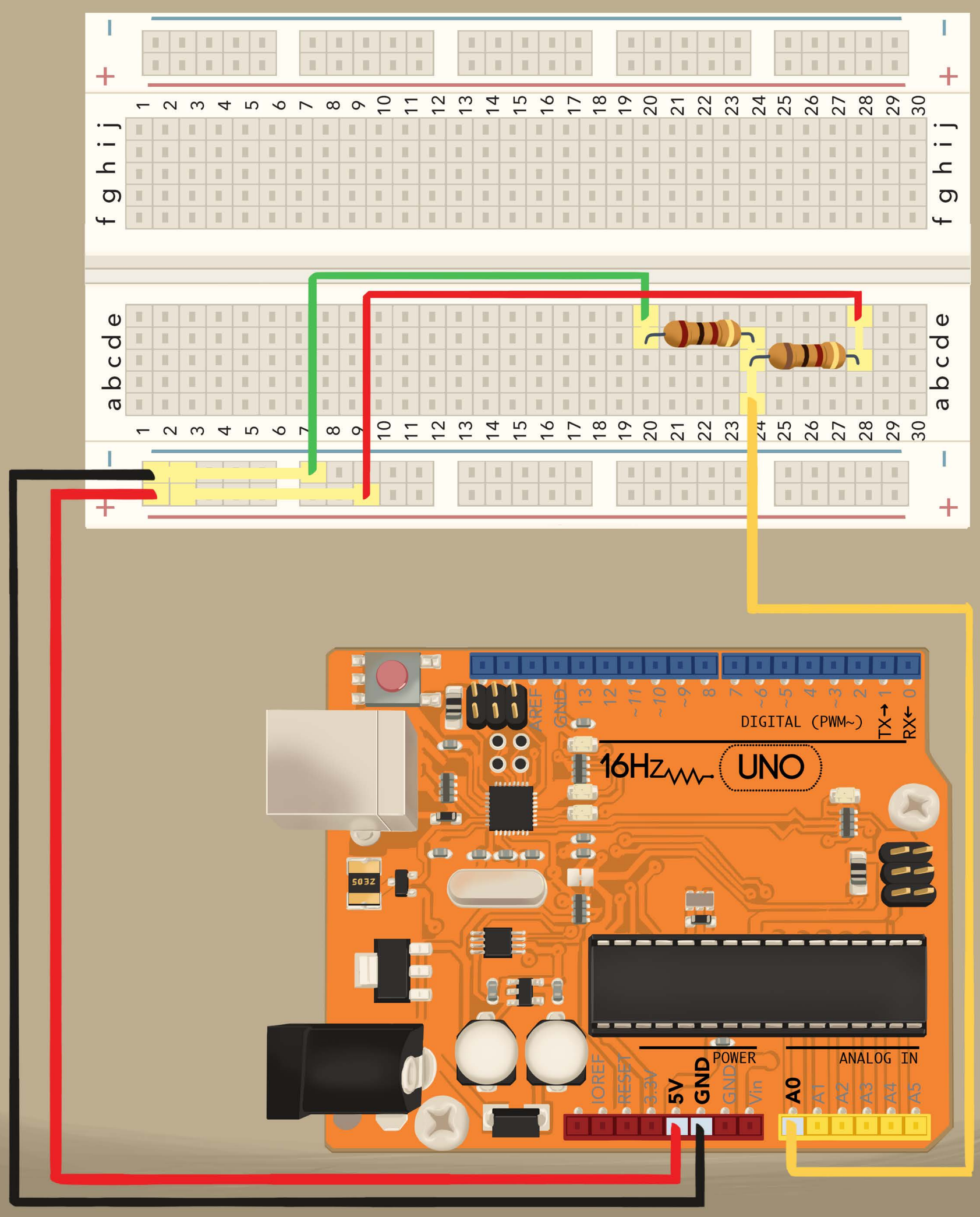


Schematic

D4.1

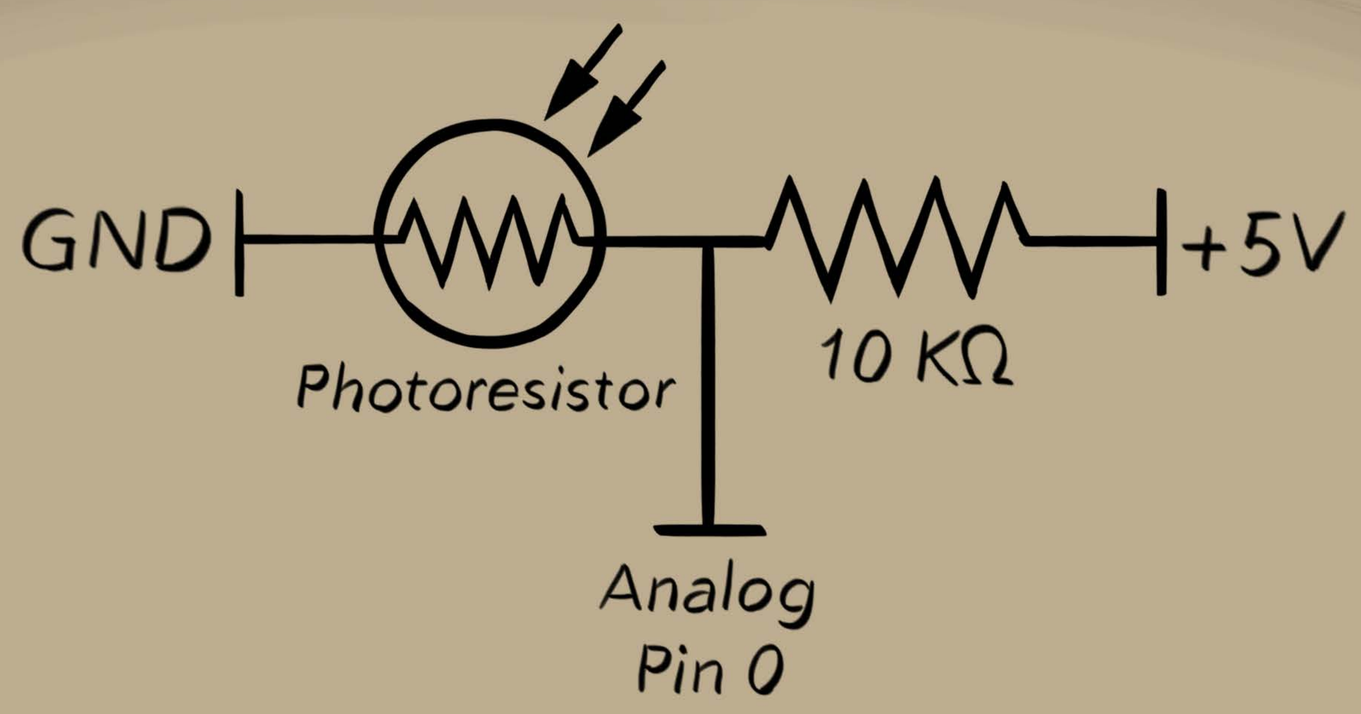


Wiring Diagram

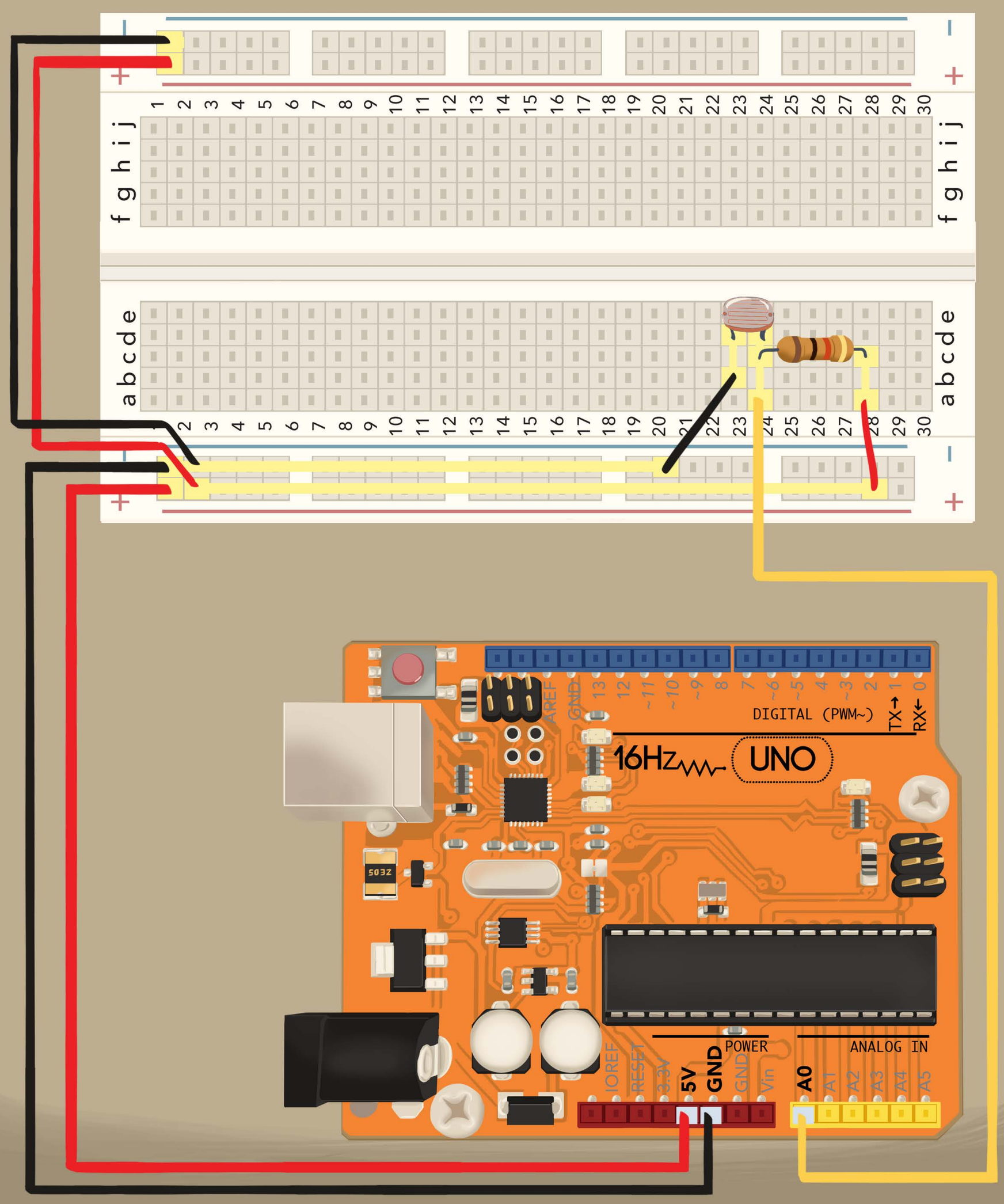


Schematic

D4.2

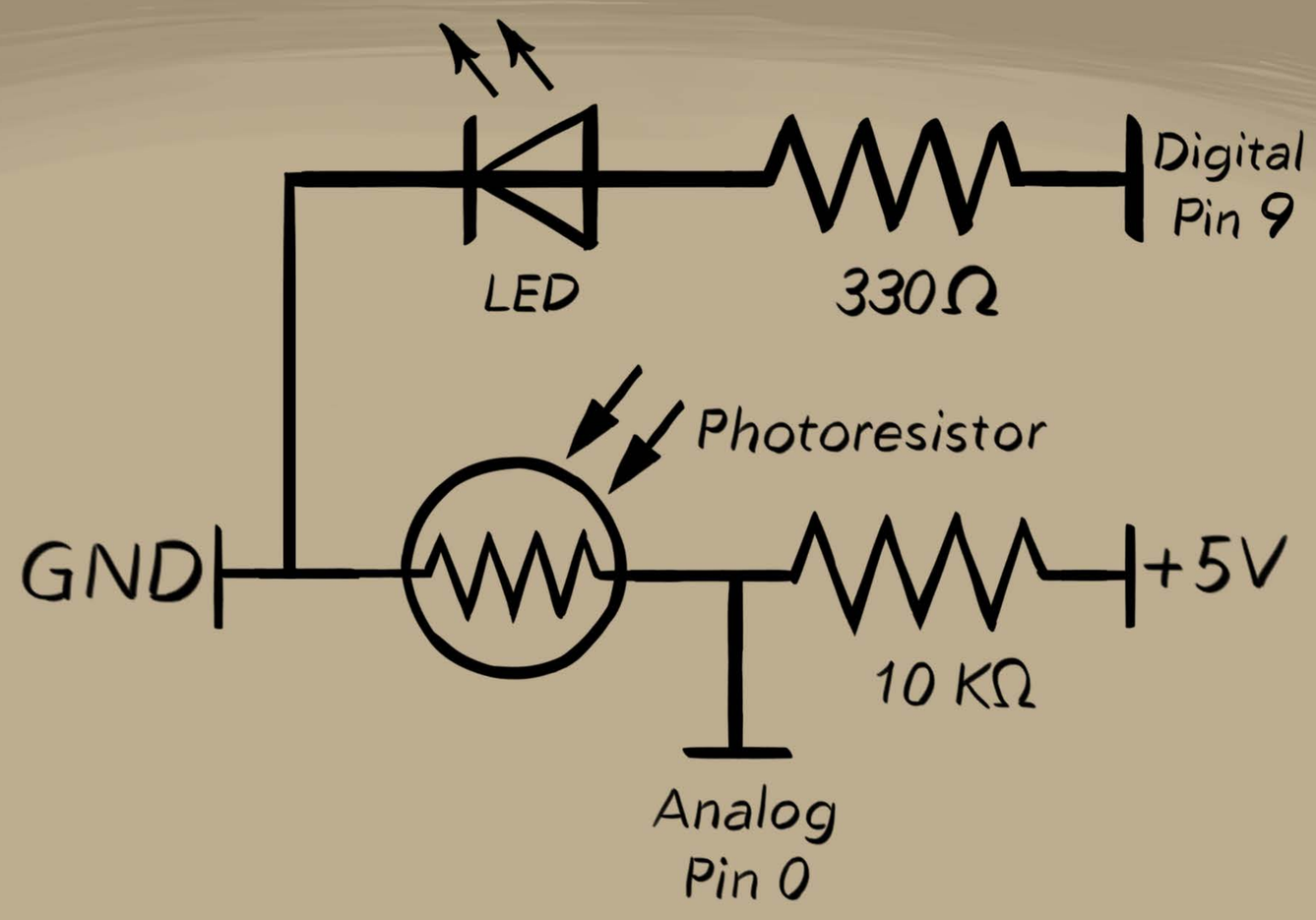


Wiring Diagram

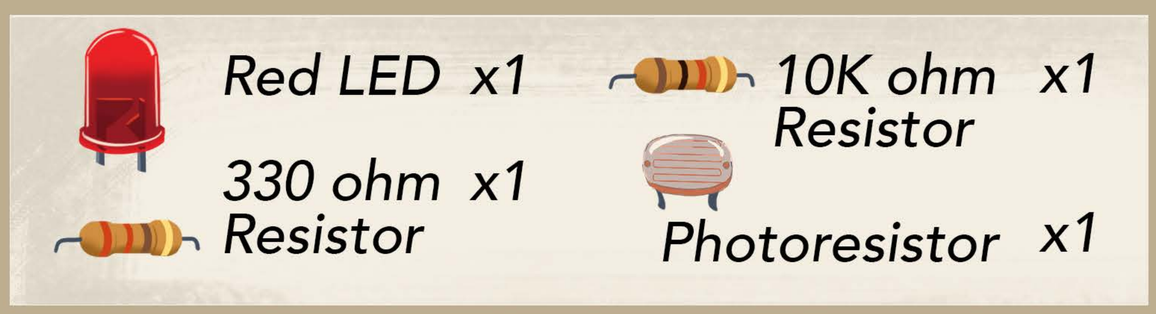


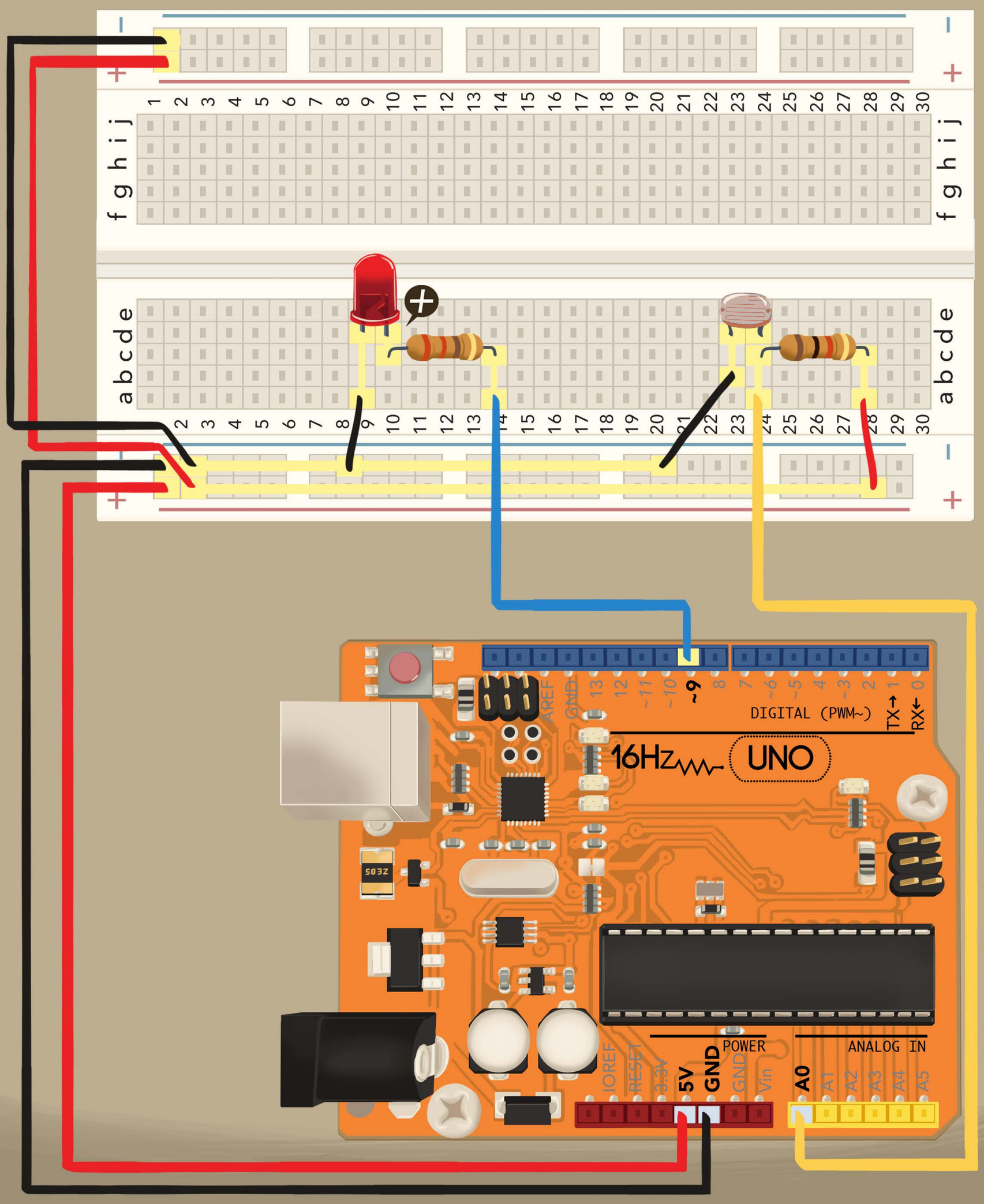
Schematic

D4.3



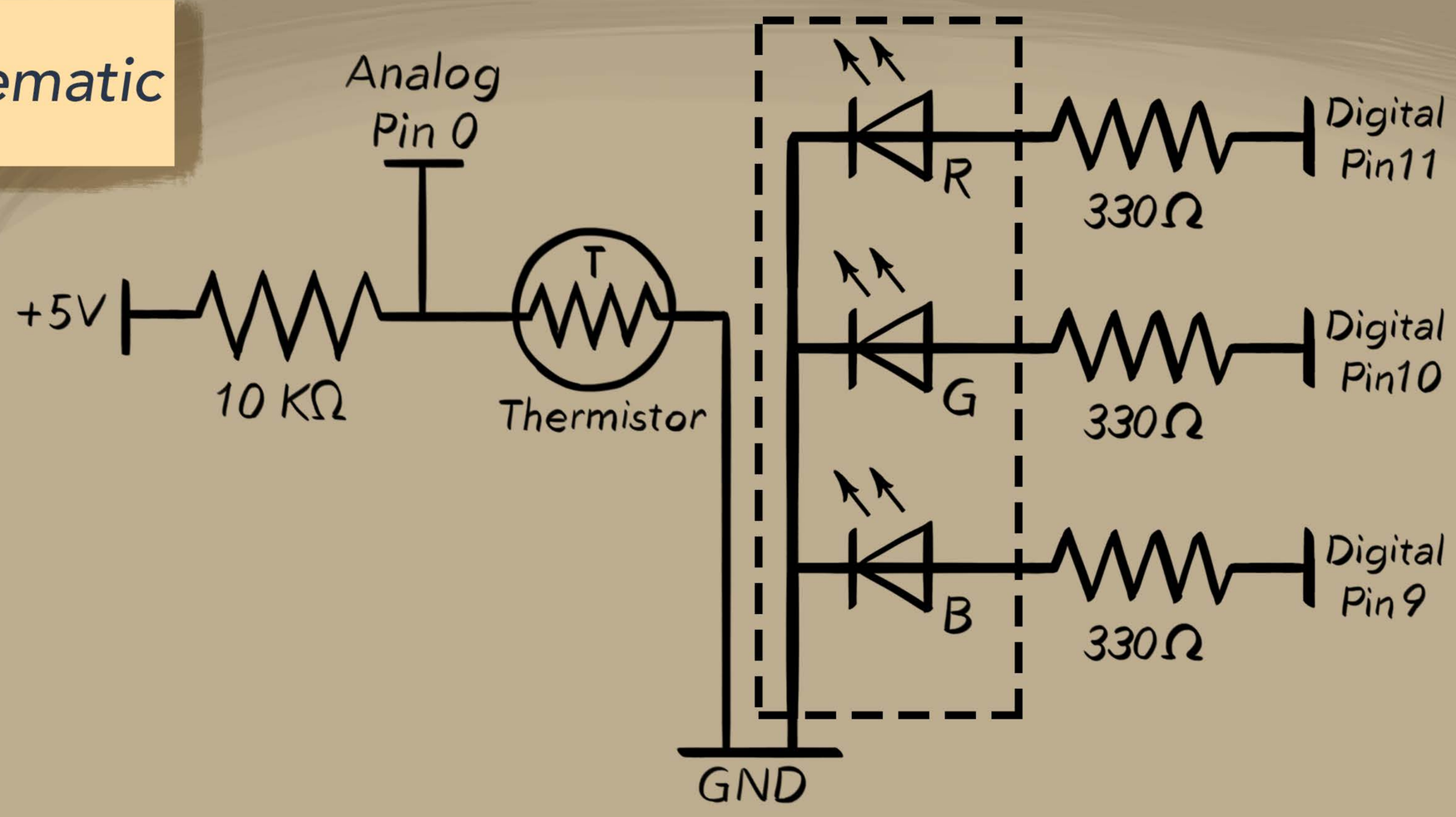
Wiring Diagram



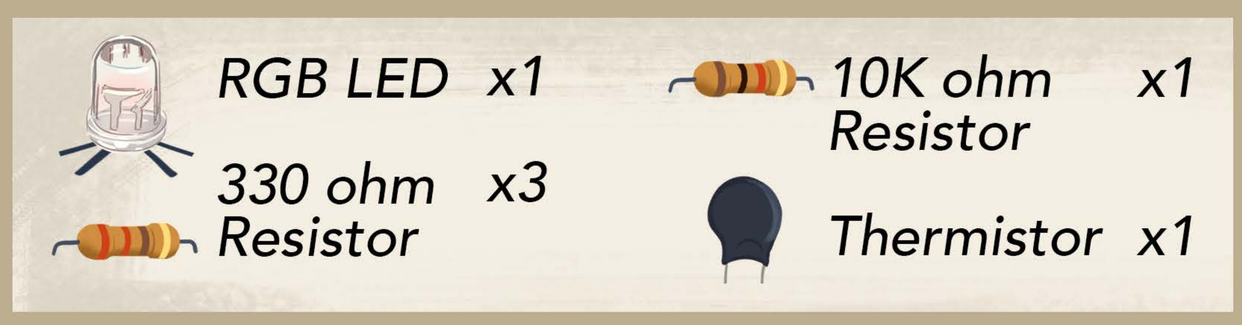


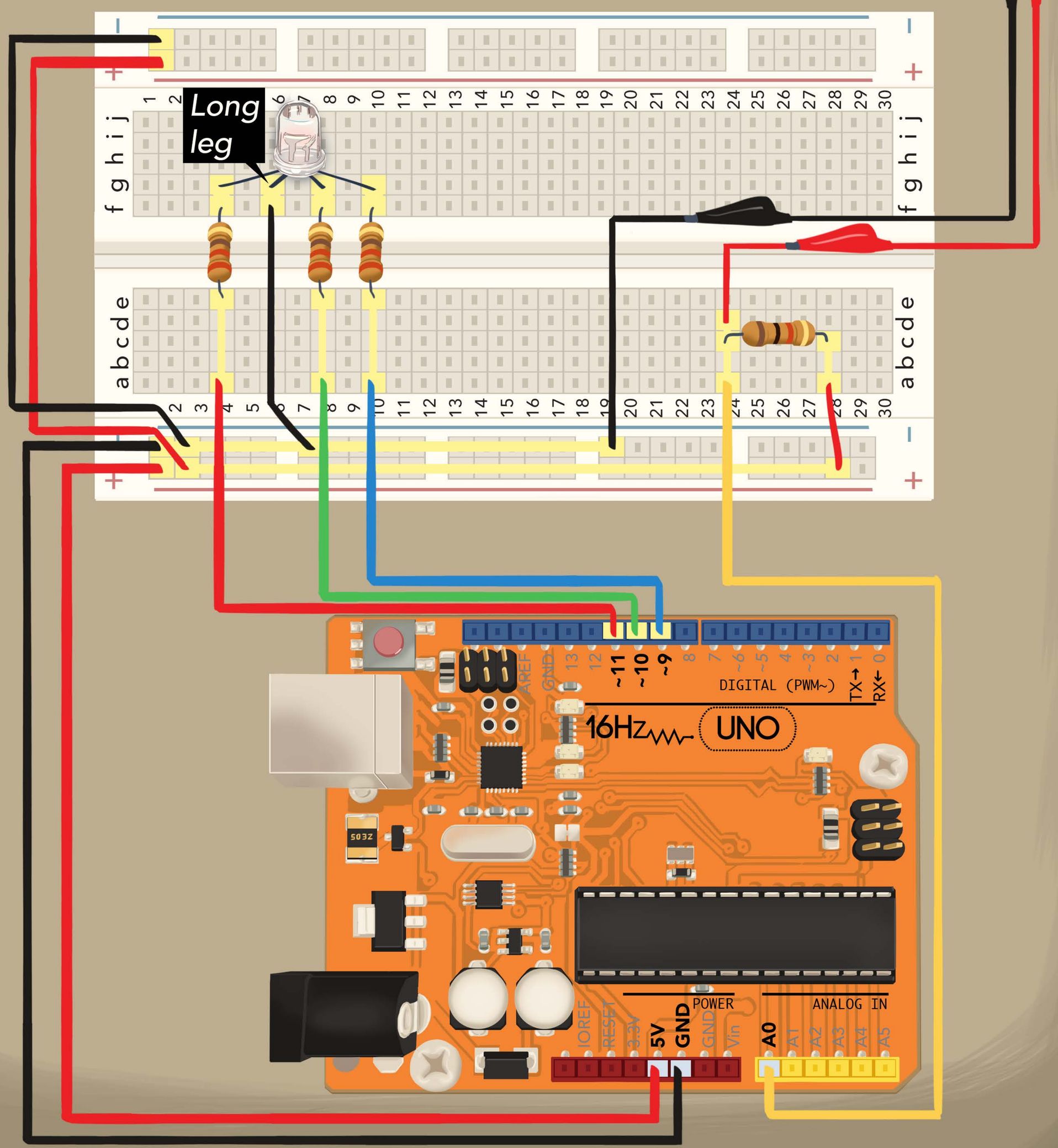
Schematic

D4.4



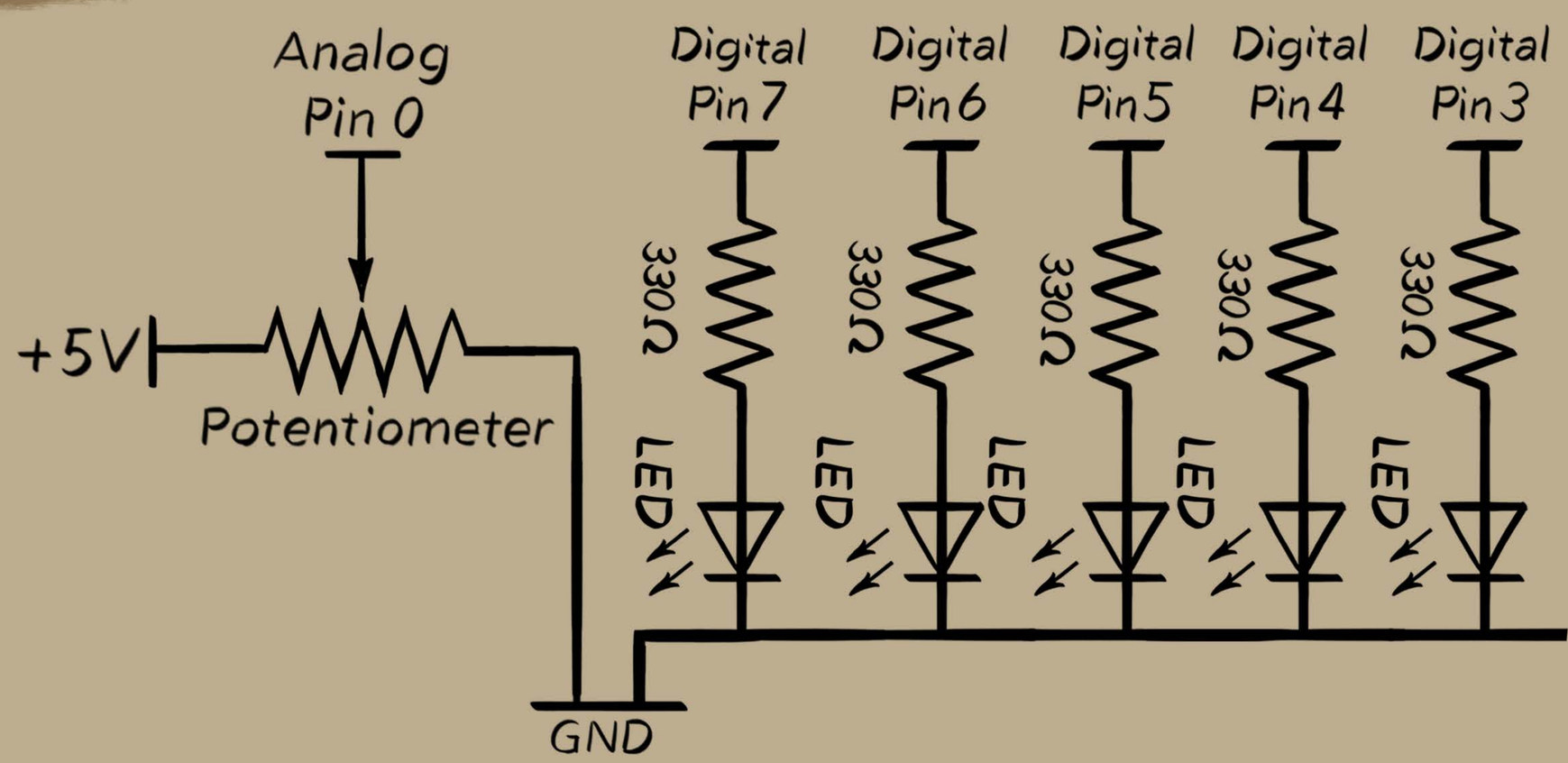
Wiring Diagram



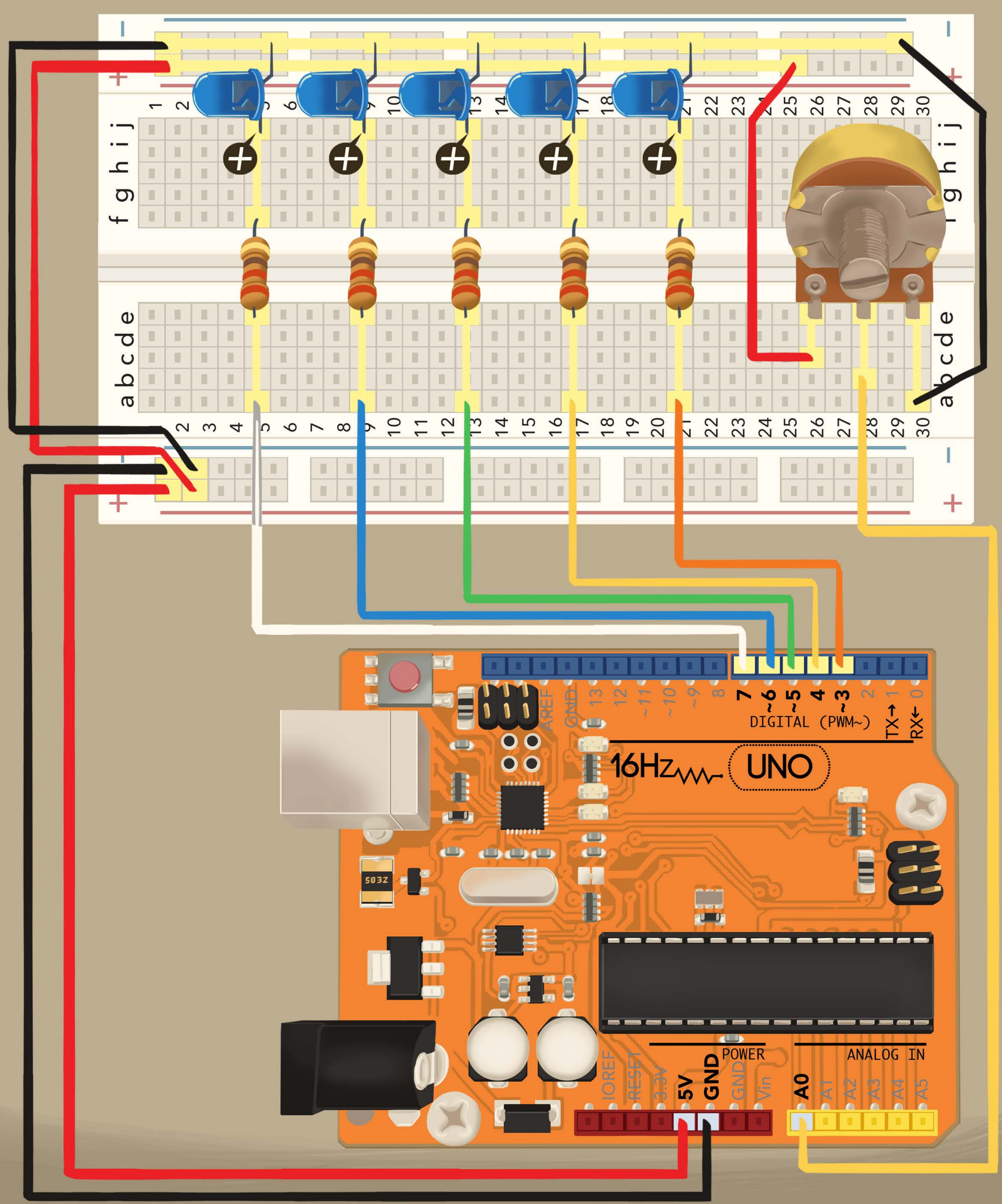


Schematic

D4.5



Wiring Diagram



ABOUT THE AUTHORS

Aditya Kumarakrishnan
Co-Author
Aditya is a co-founder at 16Hertz and fearlessly leads its product development team. With a background in applied physics and mathematics, he has undying passion for making and education. Having taught an array of subjects from robotics to astronomy in a variety of settings both in K-12 and in higher education, you can now find him mentoring high-school robotics teams and trying to develop sustainable ways schools can foster hands-on, inquiry based STEAM education.

Adiel Fernandez
Co-Author
Adiel is a lifelong learner and the Creative Director of 16Hertz. After studying physics and secondary education at the City College of New York, he taught informal science at the New York Hall of Science to learners K-12 and beyond. After finishing a graduate degree in Design and Technology from the Parsons School of Design, Adiel now finds his passion at the intersection of design, technology and education.

John Scroggins
Illustrator
John is an aspiring artist and professional comic book illustrator. Because of his passion for illustration, he developed an early obsession for video games, comics, and animation. John counts Akira Toriyama, Bill Watterson and John Kricfalusi among his larger influences and holds Bachelor of Fine Arts from the Memphis College of Art

16Hz™

CREATE SOMETHING

Want to get into the awesome world of DIY electronics and programming? The Arduino microcontroller and programming environment is a great place to start for beginners and experts alike.

This beautifully illustrated Arduino guide is the first of its kind and is designed take you through the basics of electronics and programming. Created by experienced educators, the Illustrated Arduino provides you with real-world tools and challenges coupled with fundamental electronics concepts to give you a well-rounded introduction to Arduino.

Whether you're a beginner or already familiar with microcontrollers, this guide requires little to no prior knowledge. While this guide will work for any Arduino based microcontroller, the 16Hz ULTIMATE UNO KIT *will provide you with all the parts you need to make amazing, creative projects and more!*

To see our other electronics kits and educational products from 16Hz, visit our website or follow us on social media.

16Hertz.com

@16HertzNYC

/16HertzNYC

@16HertzNYC