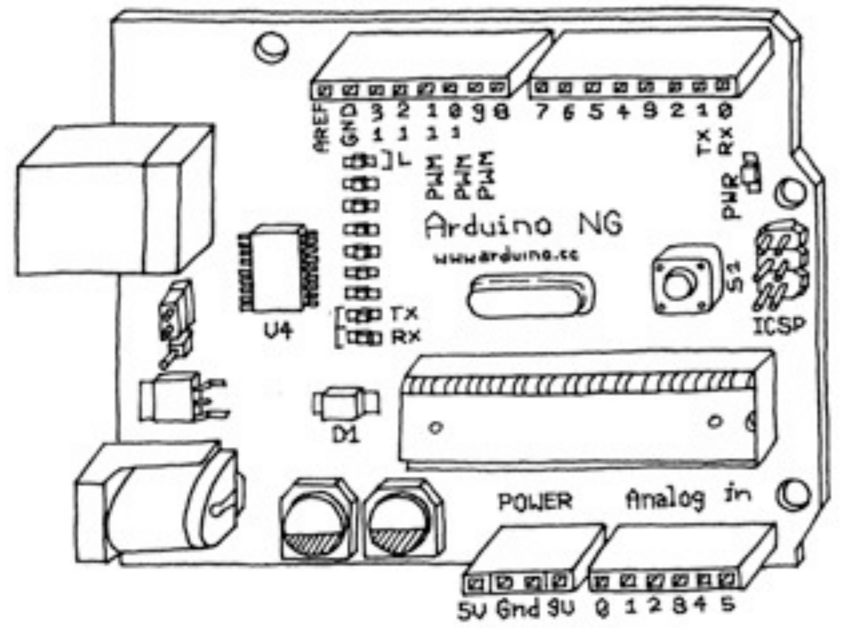
Arduino 2

Switches and Sensors

Owen Mundy | Spring 2012



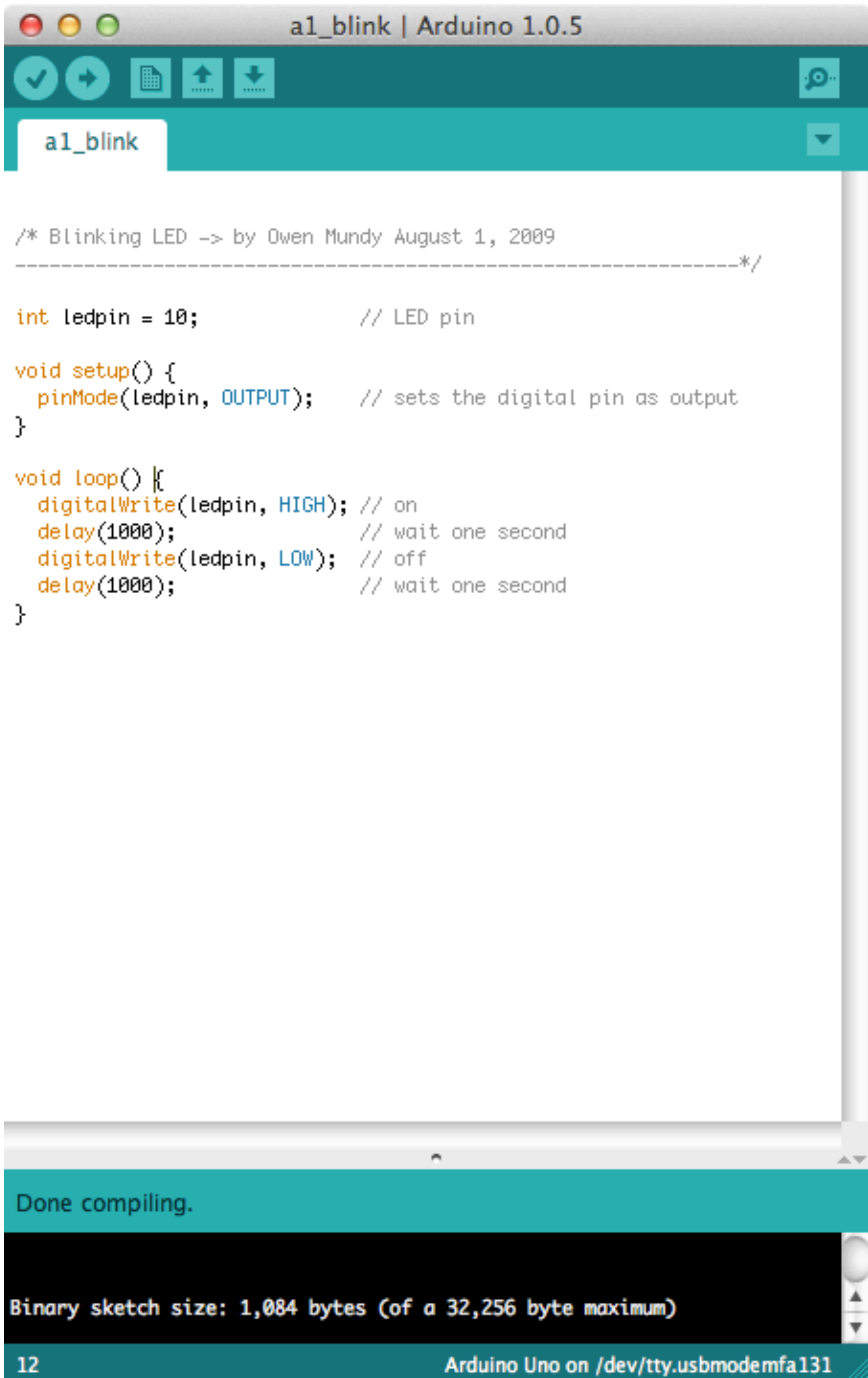
Overview

- Creating a custom function
- Serial communication
- Interactive devices: Theory of operation
- Switches
- Sensors
- Potentiometers

Creating a custom function

- Here's a trick to help keep our code clean and modular. Take our basic blink sketch.
- Every time loop() runs it executes all of the following code, turning a single LED HIGH and then LOW, and then HIGH again.

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


a1_blink | Arduino 1.0.5

a1_blink

```
/* Blinking LED -> by Owen Mundy August 1, 2009
---------------------------------------------------------------*/

int ledpin = 10;              // LED pin

void setup() {
  pinMode(ledpin, OUTPUT);    // sets the digital pin as output
}

void loop() {
  digitalWrite(ledpin, HIGH); // on
  delay(1000);                // wait one second
  digitalWrite(ledpin, LOW);  // off
  delay(1000);                // wait one second
}
```

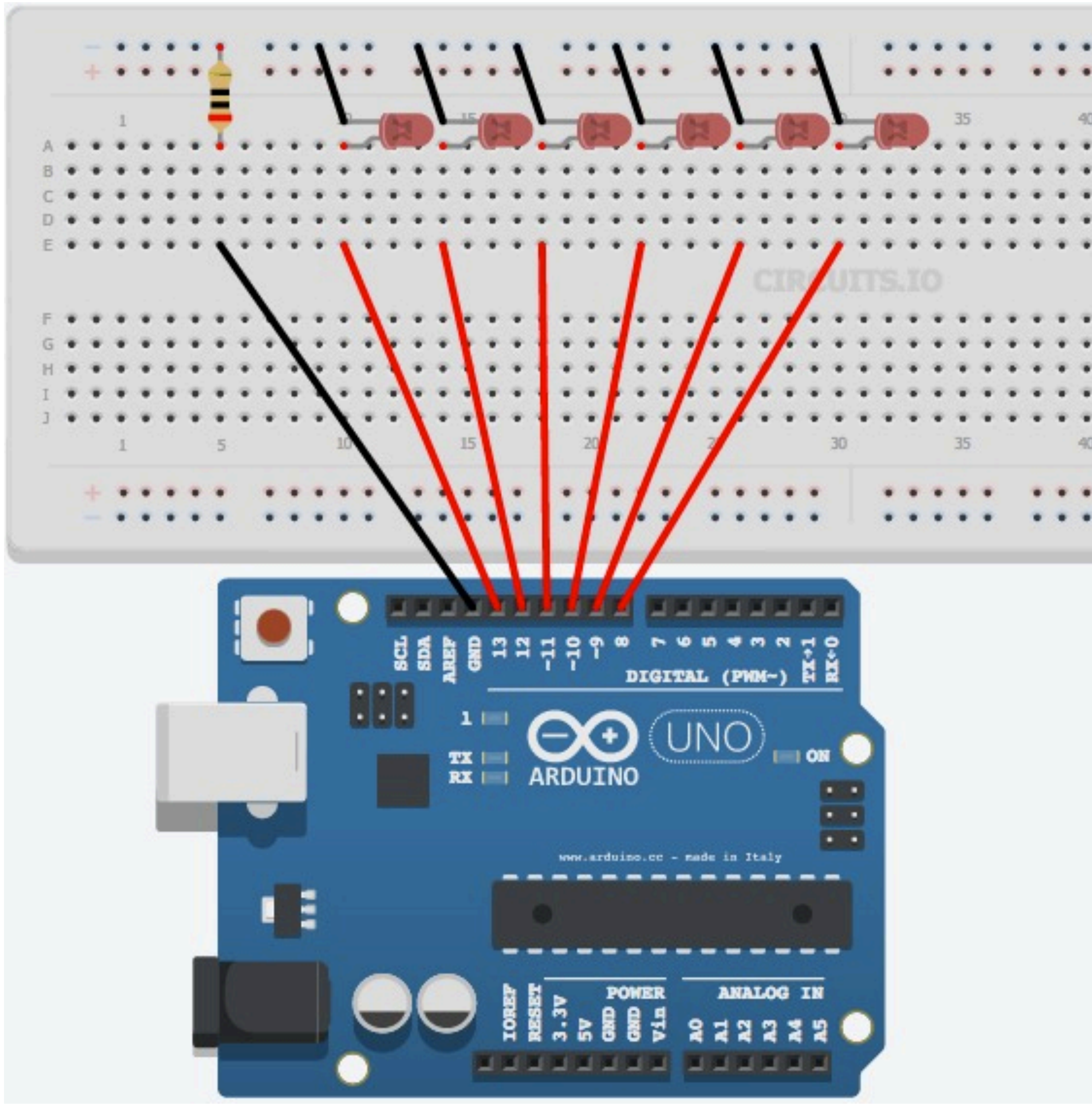
Done compiling.

```
Binary sketch size: 1,084 bytes (of a 32,256 byte maximum)
```

12 Arduino Uno on /dev/tty.usbmodemfa131

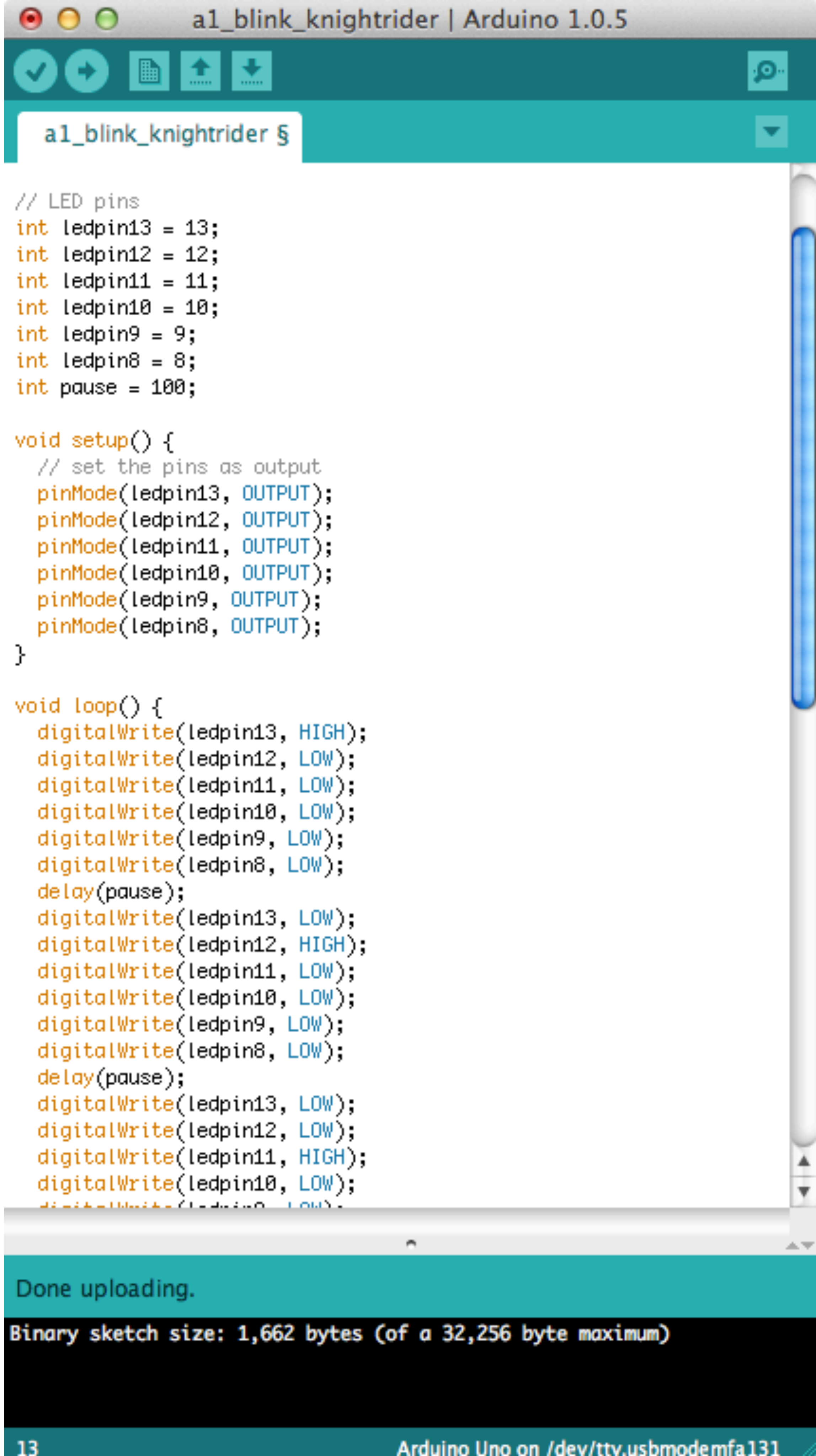
Creating a custom function

- But what if we want to control an array of LEDs individually? For example, the LEDs in this circuit?
- They all share a common ground, and the power to each is supplied by single digital pins. Very simple.
- The challenge is organizing the code...



Creating a custom function

- So using the previous “procedural” coding model from “blink” we see now the code gets quite unruly. This is because we have multiplied the lines of code * 6 (the number of LEDs).
- What if we wanted to use 20 or even 100 LEDs? That would create a real coding nightmare and possibly retinal damage!



```
// LED pins
int ledpin13 = 13;
int ledpin12 = 12;
int ledpin11 = 11;
int ledpin10 = 10;
int ledpin9 = 9;
int ledpin8 = 8;
int pause = 100;

void setup() {
  // set the pins as output
  pinMode(ledpin13, OUTPUT);
  pinMode(ledpin12, OUTPUT);
  pinMode(ledpin11, OUTPUT);
  pinMode(ledpin10, OUTPUT);
  pinMode(ledpin9, OUTPUT);
  pinMode(ledpin8, OUTPUT);
}

void loop() {
  digitalWrite(ledpin13, HIGH);
  digitalWrite(ledpin12, LOW);
  digitalWrite(ledpin11, LOW);
  digitalWrite(ledpin10, LOW);
  digitalWrite(ledpin9, LOW);
  digitalWrite(ledpin8, LOW);
  delay(pause);
  digitalWrite(ledpin13, LOW);
  digitalWrite(ledpin12, HIGH);
  digitalWrite(ledpin11, LOW);
  digitalWrite(ledpin10, LOW);
  digitalWrite(ledpin9, LOW);
  digitalWrite(ledpin8, LOW);
  delay(pause);
  digitalWrite(ledpin13, LOW);
  digitalWrite(ledpin12, LOW);
  digitalWrite(ledpin11, HIGH);
  digitalWrite(ledpin10, LOW);
```

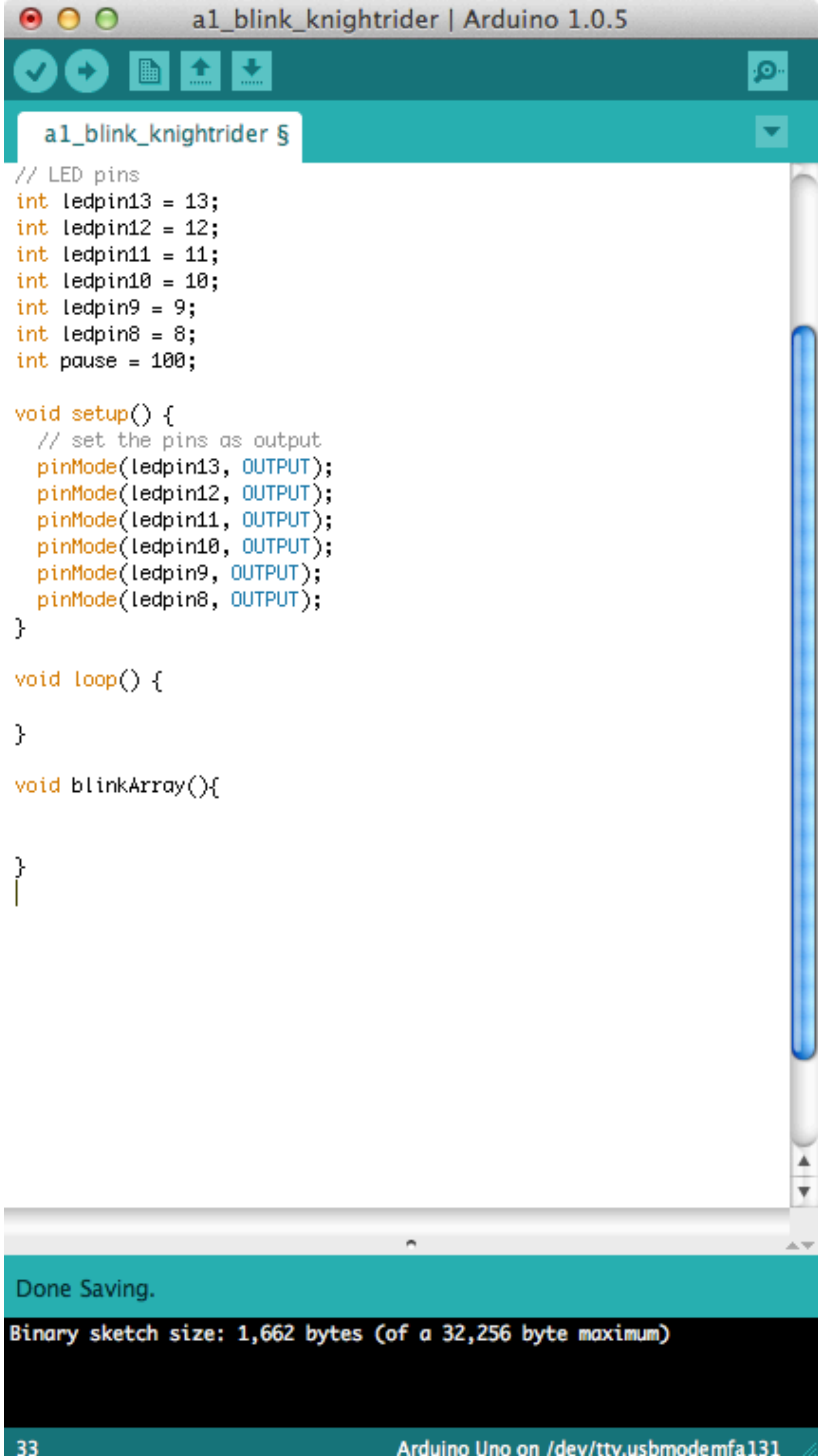
Done uploading.

```
Binary sketch size: 1,662 bytes (of a 32,256 byte maximum)
```

Creating a custom function

- So, as the name of this section suggests. We will create a function that can be reused over and over. This will allow us the ability to make big changes to our circuit without having to rewrite 1000s of lines of code.
- Start by creating the name and wrapping curly brackets:

```
void blinkArray(){

}
```



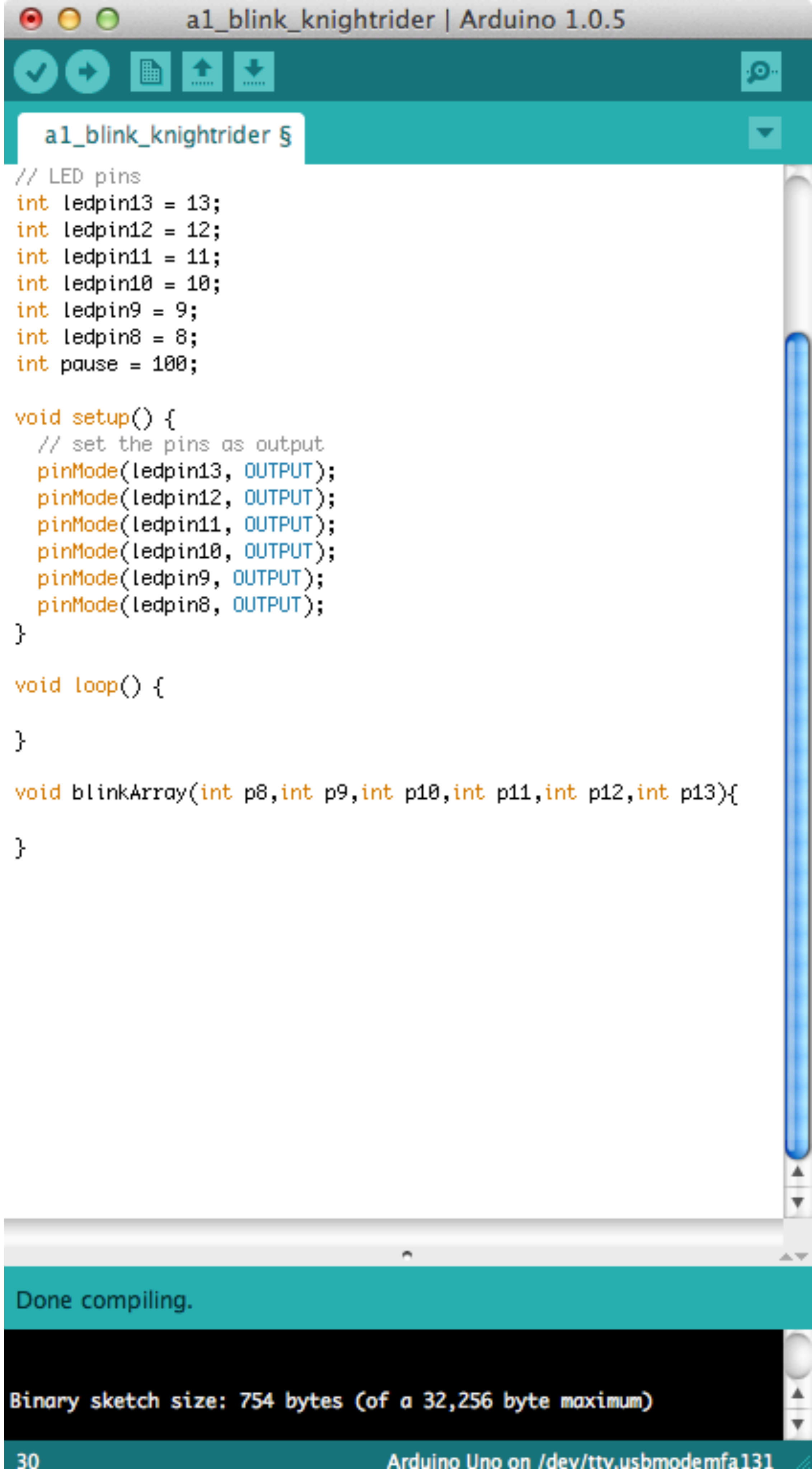
```
// LED pins
int ledpin13 = 13;
int ledpin12 = 12;
int ledpin11 = 11;
int ledpin10 = 10;
int ledpin9 = 9;
int ledpin8 = 8;
int pause = 100;

void setup() {
  // set the pins as output
  pinMode(ledpin13, OUTPUT);
  pinMode(ledpin12, OUTPUT);
  pinMode(ledpin11, OUTPUT);
  pinMode(ledpin10, OUTPUT);
  pinMode(ledpin9, OUTPUT);
  pinMode(ledpin8, OUTPUT);
}

void loop() {

}

void blinkArray(){


}
```

```
Done Saving.
Binary sketch size: 1,662 bytes (of a 32,256 byte maximum)
```

Creating a custom function

- Now we add parameters to the function to receive data when we call it. Each of these will be an integer or int.

```
void blinkArray(int p8,int p9,int p10,
                int p11,int p12,int p13){

}
```



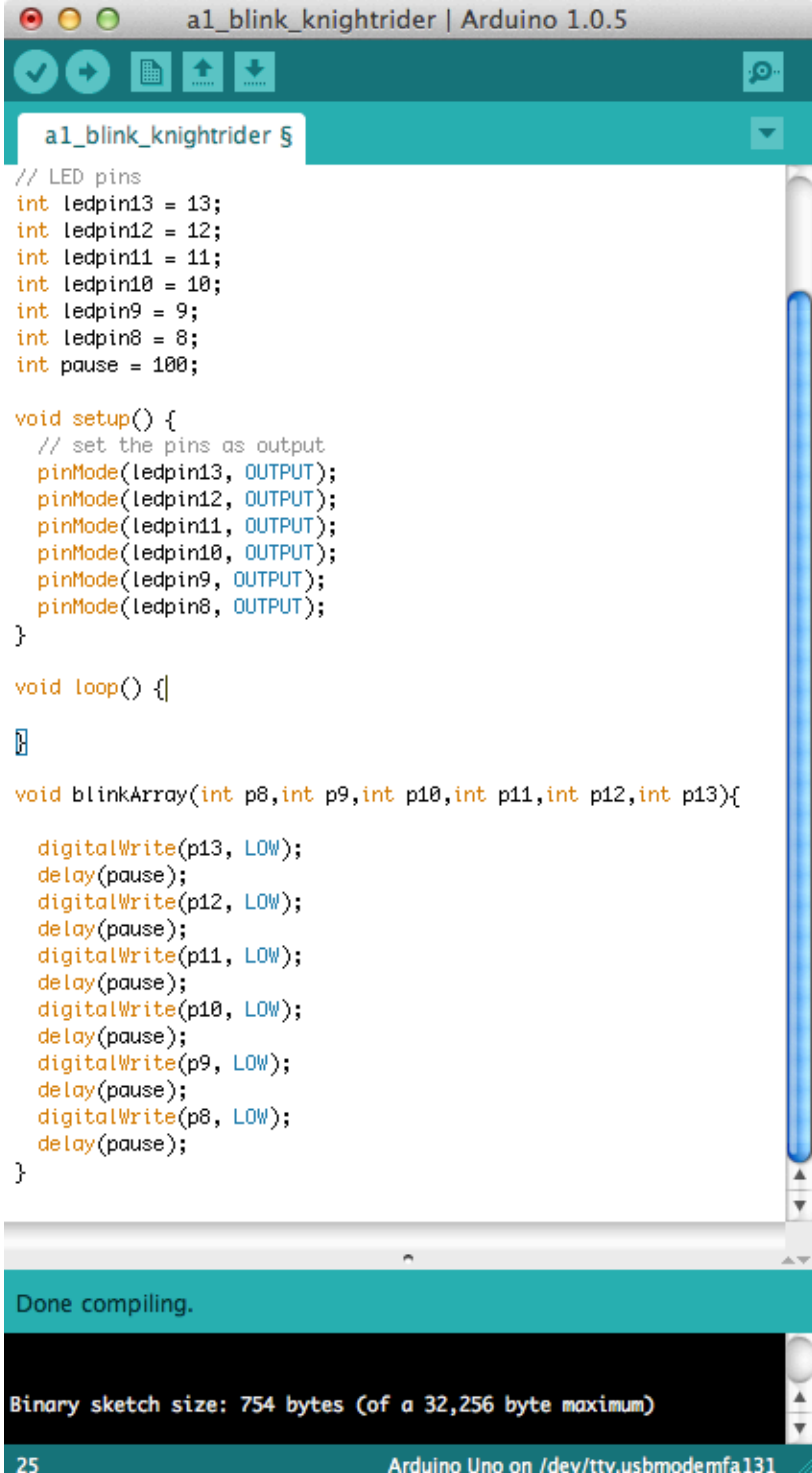
```
// LED pins
int ledpin13 = 13;
int ledpin12 = 12;
int ledpin11 = 11;
int ledpin10 = 10;
int ledpin9 = 9;
int ledpin8 = 8;
int pause = 100;

void setup() {
  // set the pins as output
  pinMode(ledpin13, OUTPUT);
  pinMode(ledpin12, OUTPUT);
  pinMode(ledpin11, OUTPUT);
  pinMode(ledpin10, OUTPUT);
  pinMode(ledpin9, OUTPUT);
  pinMode(ledpin8, OUTPUT);
}

void loop() {

}

void blinkArray(int p8,int p9,int p10,int p11,int p12,int p13){

}
```

```
Done compiling.

Binary sketch size: 754 bytes (of a 32,256 byte maximum)
```

Creating a custom function

- Then we use digitalWrite() to set each LED HIGH or LOW by referencing the pin number (e.g. ledpin13) and the value that pin should be changed to (e.g. stored in p13).

```
void blinkArray(int p8,int p9,int p10,
                            int p11,int p12,int p13){
  digitalWrite(ledpin13, p13);
  digitalWrite(ledpin12, p12);
  digitalWrite(ledpin11, p11);
  digitalWrite(ledpin10, p10);
  digitalWrite(ledpin9, p9);
  digitalWrite(ledpin8, p8);
  delay(pause);
}
```



```
// LED pins
int ledpin13 = 13;
int ledpin12 = 12;
int ledpin11 = 11;
int ledpin10 = 10;
int ledpin9 = 9;
int ledpin8 = 8;
int pause = 100;

void setup() {
  // set the pins as output
  pinMode(ledpin13, OUTPUT);
  pinMode(ledpin12, OUTPUT);
  pinMode(ledpin11, OUTPUT);
  pinMode(ledpin10, OUTPUT);
  pinMode(ledpin9, OUTPUT);
  pinMode(ledpin8, OUTPUT);
}

void loop() {

}

void blinkArray(int p8,int p9,int p10,int p11,int p12,int p13){

  digitalWrite(p13, LOW);
  delay(pause);
  digitalWrite(p12, LOW);
  delay(pause);
  digitalWrite(p11, LOW);
  delay(pause);
  digitalWrite(p10, LOW);
  delay(pause);
  digitalWrite(p9, LOW);
  delay(pause);
  digitalWrite(p8, LOW);
  delay(pause);
}
```

```
Done compiling.

Binary sketch size: 754 bytes (of a 32,256 byte maximum)
```

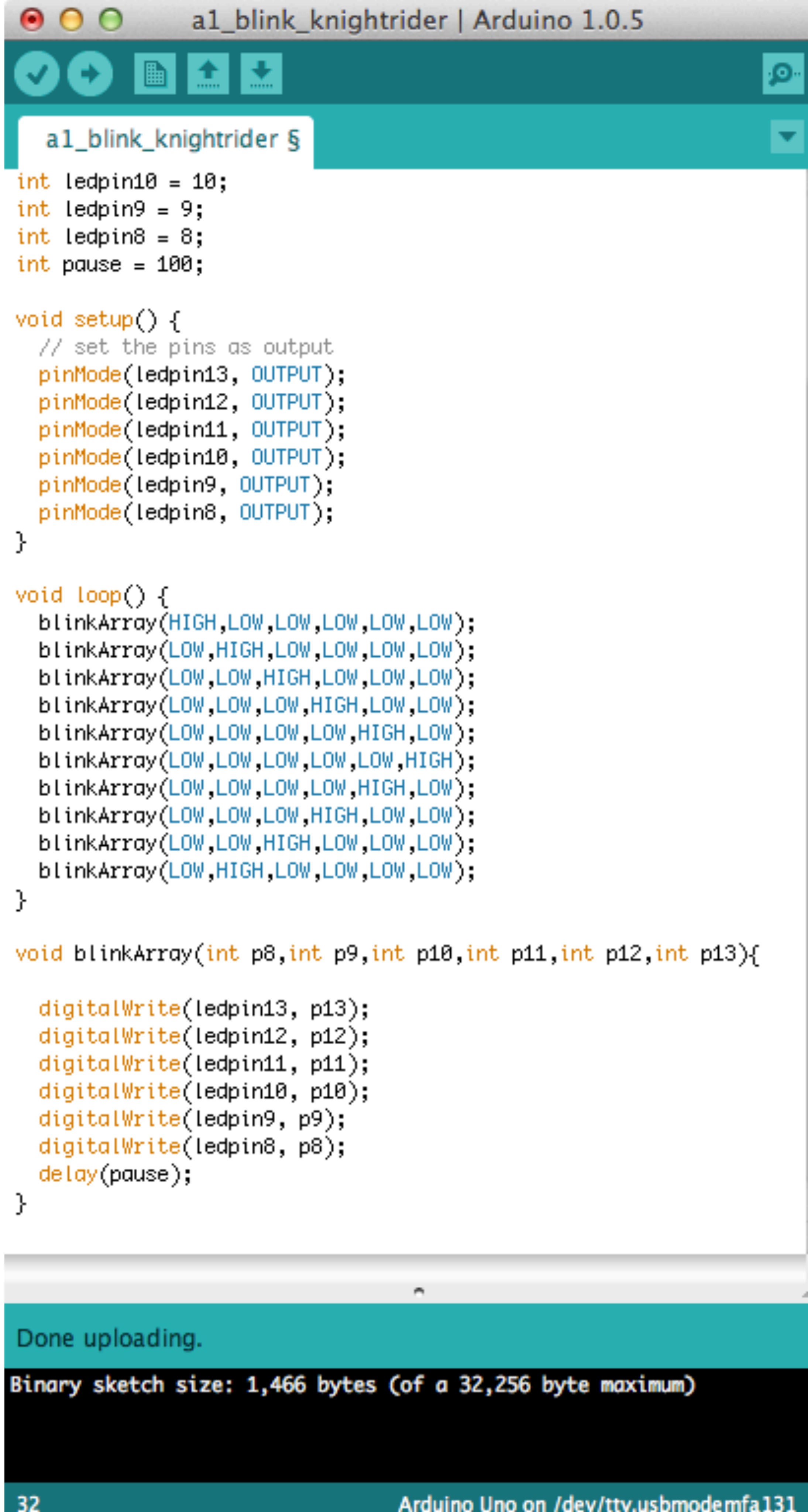
Creating a custom function

- Now we call the function six times, each time passing the values we want all the LEDs in the array to assume.

```
void loop() {
  blinkArray(HIGH,LOW,LOW,LOW,LOW,LOW);
  blinkArray(LOW,HIGH,LOW,LOW,LOW,LOW);
  blinkArray(LOW,LOW,HIGH,LOW,LOW,LOW);
  ...
}
```

Voila!! http://123d.circuits.io/circuits/37068

- Question: How could we make the code even more efficient? Hint: The word we used to describe the group of six LEDs...



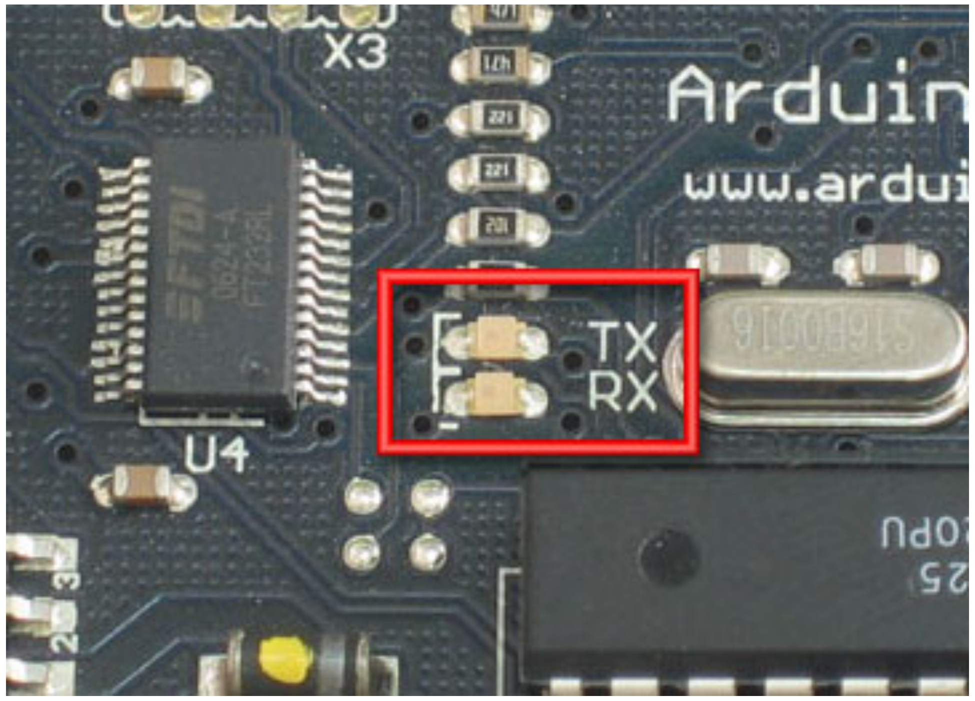
```
int ledpin10 = 10;
int ledpin9 = 9;
int ledpin8 = 8;
int pause = 100;

void setup() {
  // set the pins as output
  pinMode(ledpin13, OUTPUT);
  pinMode(ledpin12, OUTPUT);
  pinMode(ledpin11, OUTPUT);
  pinMode(ledpin10, OUTPUT);
  pinMode(ledpin9, OUTPUT);
  pinMode(ledpin8, OUTPUT);
}

void loop() {
  blinkArray(HIGH,LOW,LOW,LOW,LOW,LOW);
  blinkArray(LOW,HIGH,LOW,LOW,LOW,LOW);
  blinkArray(LOW,LOW,HIGH,LOW,LOW,LOW);
  blinkArray(LOW,LOW,LOW,HIGH,LOW,LOW);
  blinkArray(LOW,LOW,LOW,LOW,HIGH,LOW);
  blinkArray(LOW,LOW,LOW,LOW,LOW,HIGH);
  blinkArray(LOW,LOW,LOW,LOW,HIGH,LOW);
  blinkArray(LOW,LOW,LOW,HIGH,LOW,LOW);
  blinkArray(LOW,LOW,HIGH,LOW,LOW,LOW);
  blinkArray(LOW,HIGH,LOW,LOW,LOW,LOW);
}

void blinkArray(int p8,int p9,int p10,int p11,int p12,int p13){

  digitalWrite(ledpin13, p13);
  digitalWrite(ledpin12, p12);
  digitalWrite(ledpin11, p11);
  digitalWrite(ledpin10, p10);
  digitalWrite(ledpin9, p9);
  digitalWrite(ledpin8, p8);
  delay(pause);
}
```

Done uploading.

```
Binary sketch size: 1,466 bytes (of a 32,256 byte maximum)
```

Serial communication

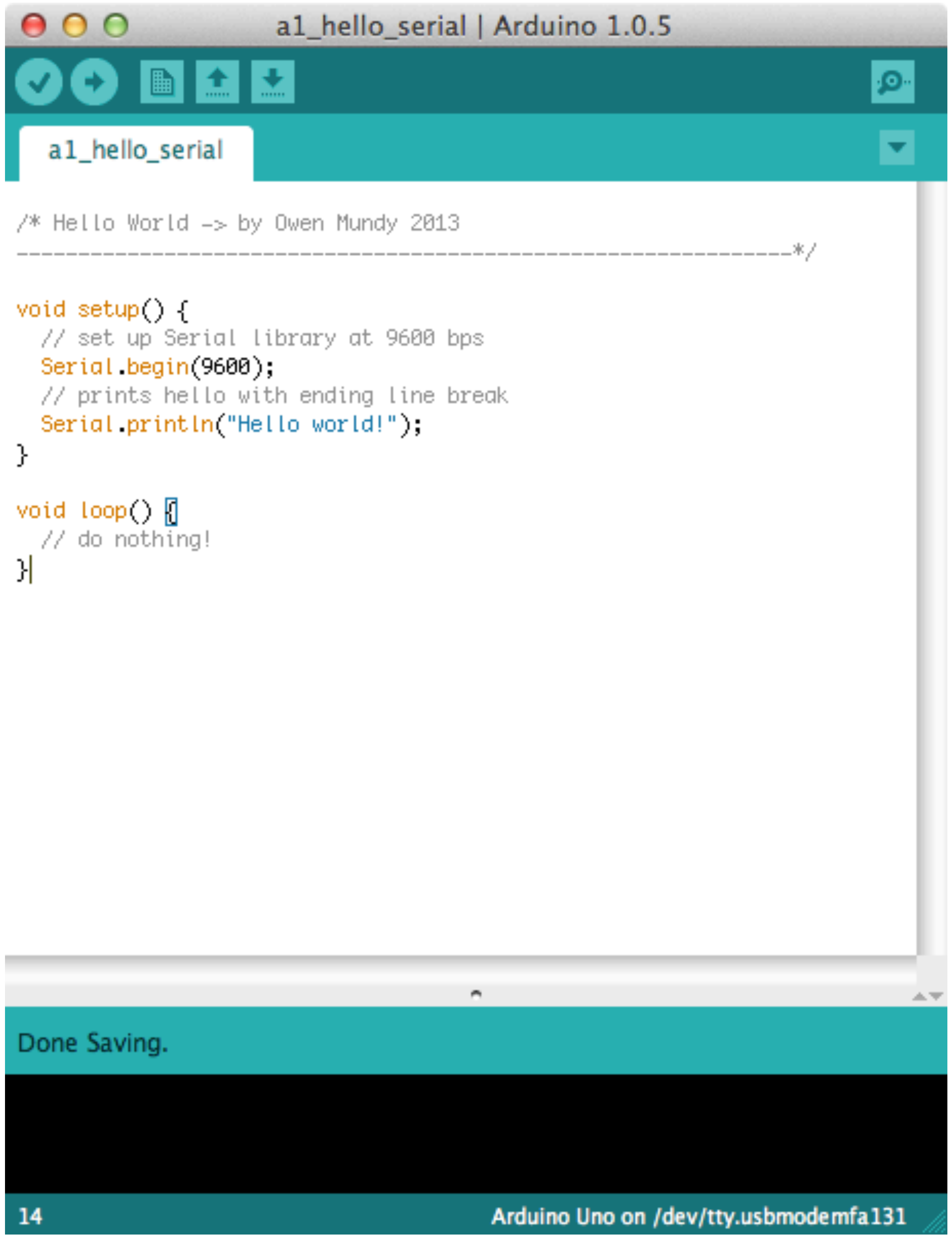
- Before we start using sensors and actuators it’s good to have an understanding of how to send data to and from your Arduino via USB.
- When we talk about serial communication, data literally moves in a series, one bit after another. It’s like really fast morse code.
- We use serial communication already. It’s how we upload sketches to our Arduino. (Technically it is first compiled into binary data, then uploaded)
- Watch the TX an RX LEDs on your Ardunio next time you upload a sketch. TX blinks when it is **T**ransmitting data, and RX when data is **R**ecieved.



Serial communication

- You can also send and receive serial message with the Arduino. Consider the following code:

```
void setup() {
  // set up Serial library at 9600 bps
  Serial.begin(9600);
  // prints hello with ending line break
  Serial.println("Hello world!");
}
void loop() {
  // do nothing!
}
```



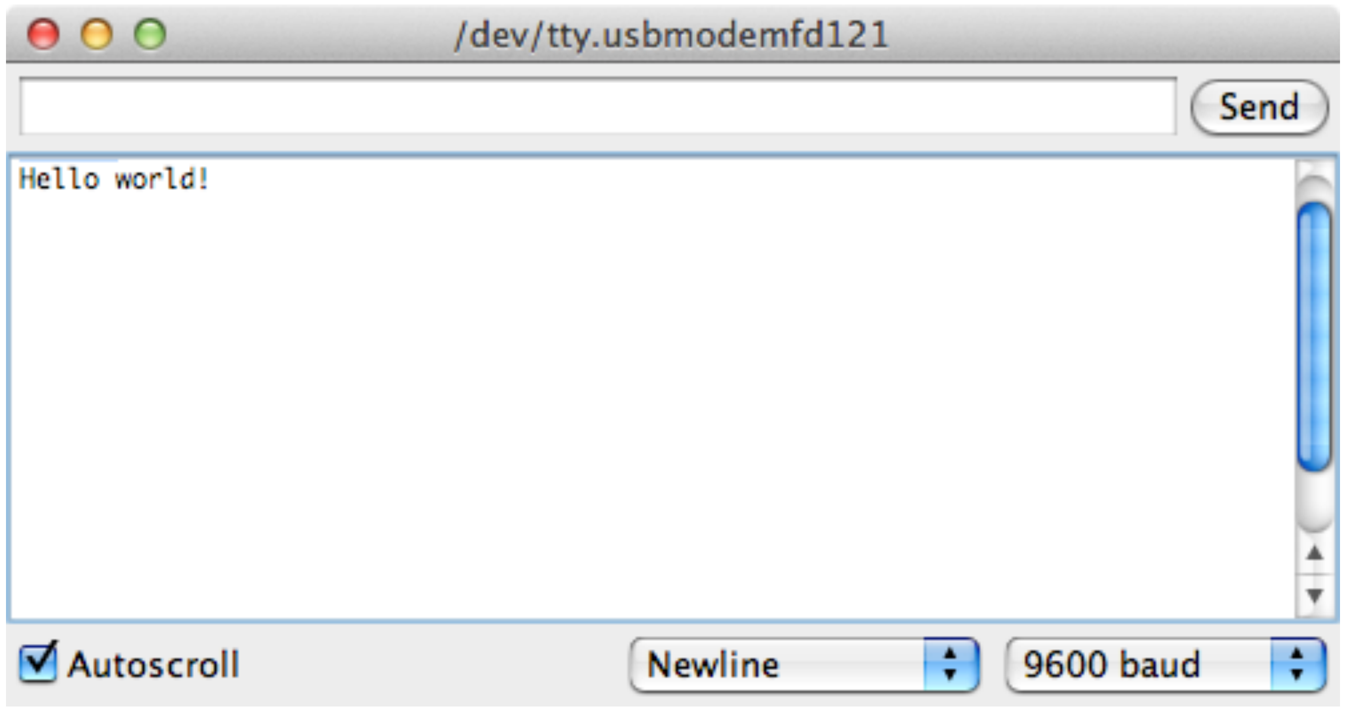
```
/* Hello World -> by Owen Mundy 2013
---------------------------------------------------------------*/

void setup() {
  // set up Serial library at 9600 bps
  Serial.begin(9600);
  // prints hello with ending line break
  Serial.println("Hello world!");
}

void loop() {
  // do nothing!
}
```

Done Saving.

14 Arduino Uno on /dev/tty.usbmodemfa131

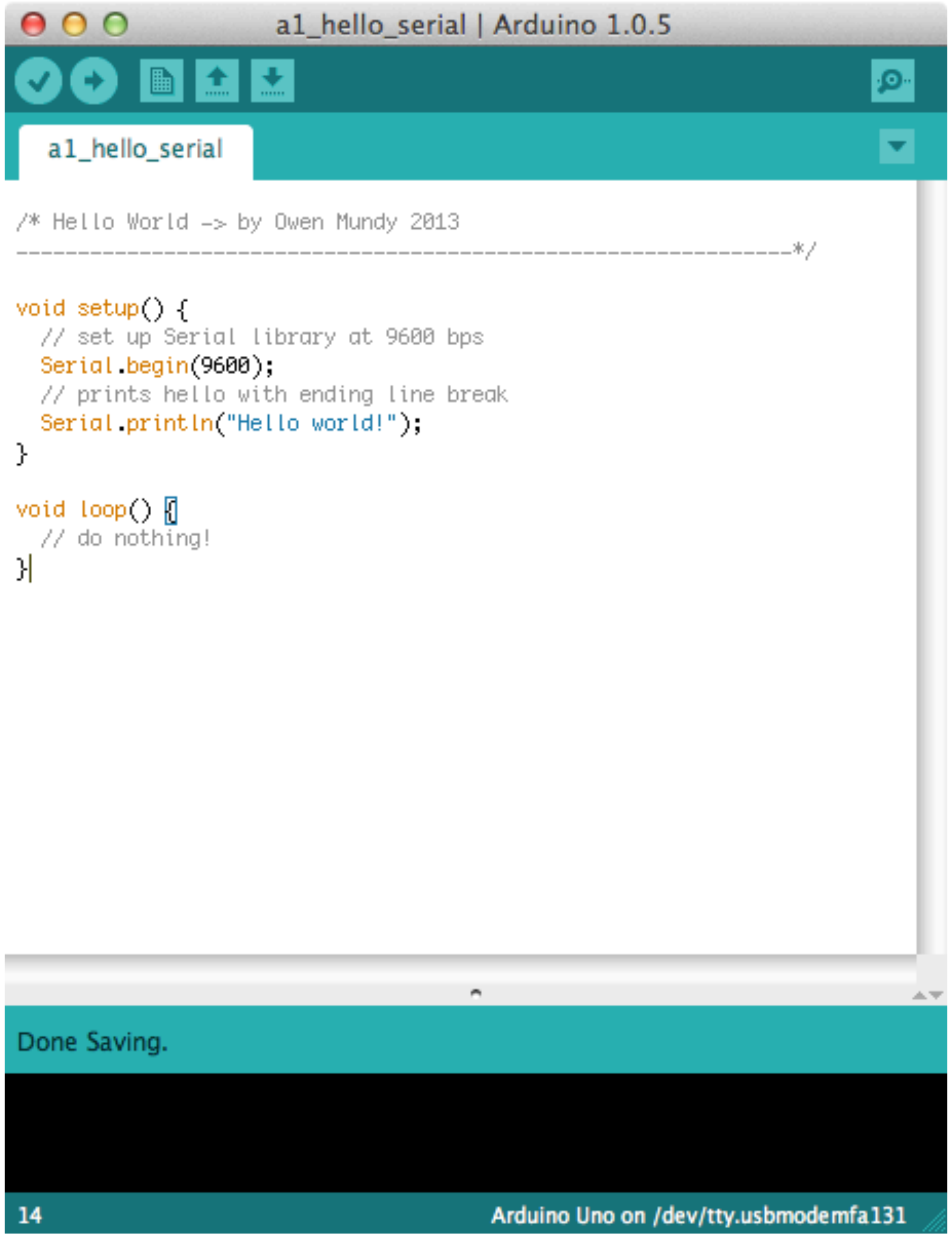
Serial communication

- Choose: Tools > Serial Monitor to see the communication back from the Ardunio.
- Note: Make sure you have the correct baud rate selected in the monitor (it should match your code).



```
Hello world!
```

Autoscroll ✓ — Newline — 9600 baud



```
/* Hello World -> by Owen Mundy 2013
-----------------------------------------------------------------*/

void setup() {
  // set up Serial library at 9600 bps
  Serial.begin(9600);
  // prints hello with ending line break
  Serial.println("Hello world!");
}

void loop() {
  // do nothing!
}
```

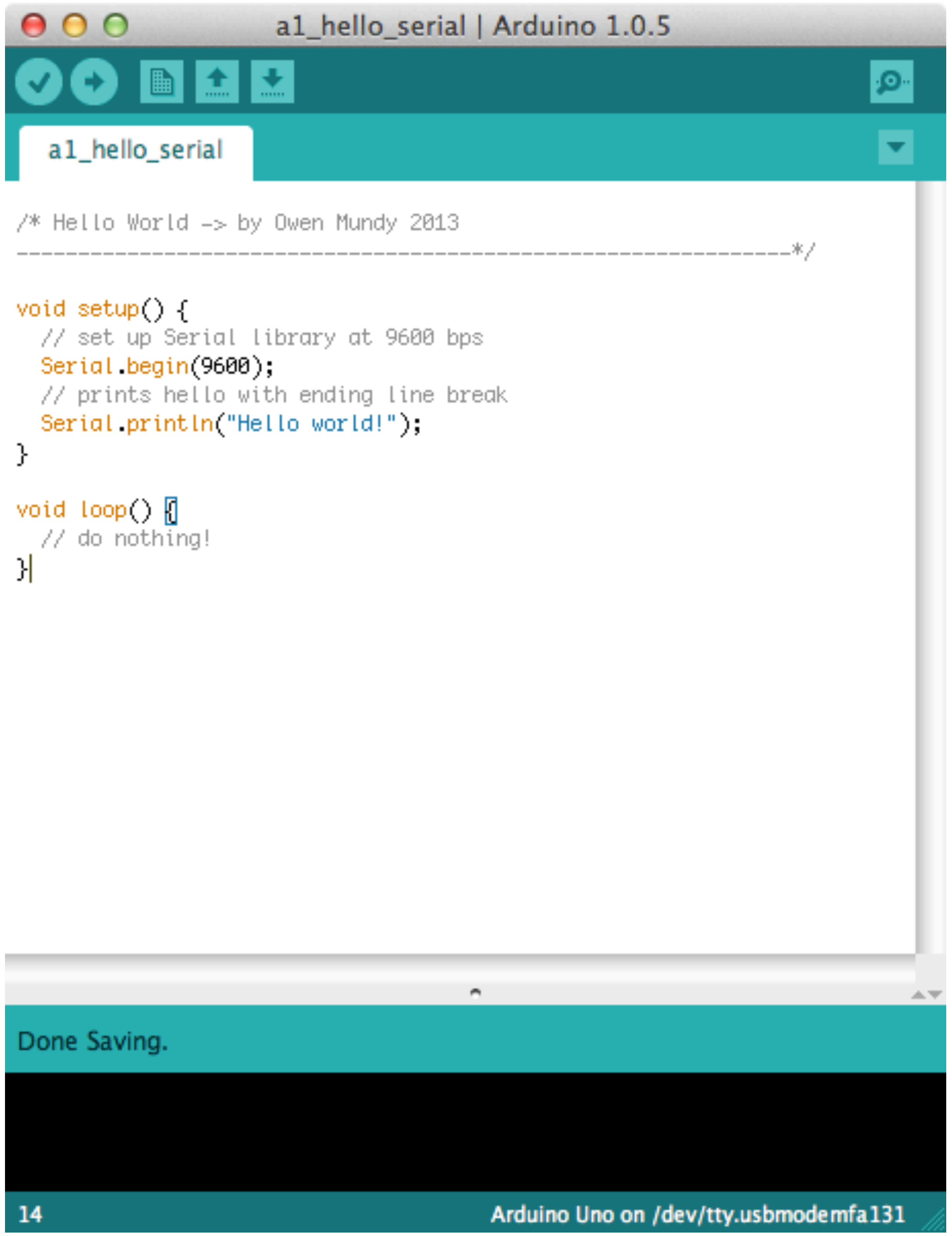
Done Saving.

14 — Arduino Uno on /dev/tty.usbmodemfa131

- So what's going on in here?

Serial communication

- In our setup() function we are using the Serial library to start the communication.
- Libraries in programming are similar to real libraries. Meaning, they contain a wealth of knowledge that can help us imagine new possibilities.
- Specifically, the Serial library gives us functions to talk with the Arduino.
- When we say Serial.begin(9600) we're starting a serial connection at 9600 bps (bits-per-second) (a.k.a. "the baud rate"). Compare this to a 56k modem the Arduino is only 9.375 Kbps or about 1/5 the speed! Still, it's fast enough for our purposes.
- Serial.println("Hello world") just tells the Arduno to print the string back to us.


a1_hello_serial | Arduino 1.0.5
a1_hello_serial

```
/* Hello World -> by Owen Mundy 2013
-----------------------------------------------------------------*/

void setup() {
  // set up Serial library at 9600 bps
  Serial.begin(9600);
  // prints hello with ending line break
  Serial.println("Hello world!");
}

void loop() {
  // do nothing!
}
```

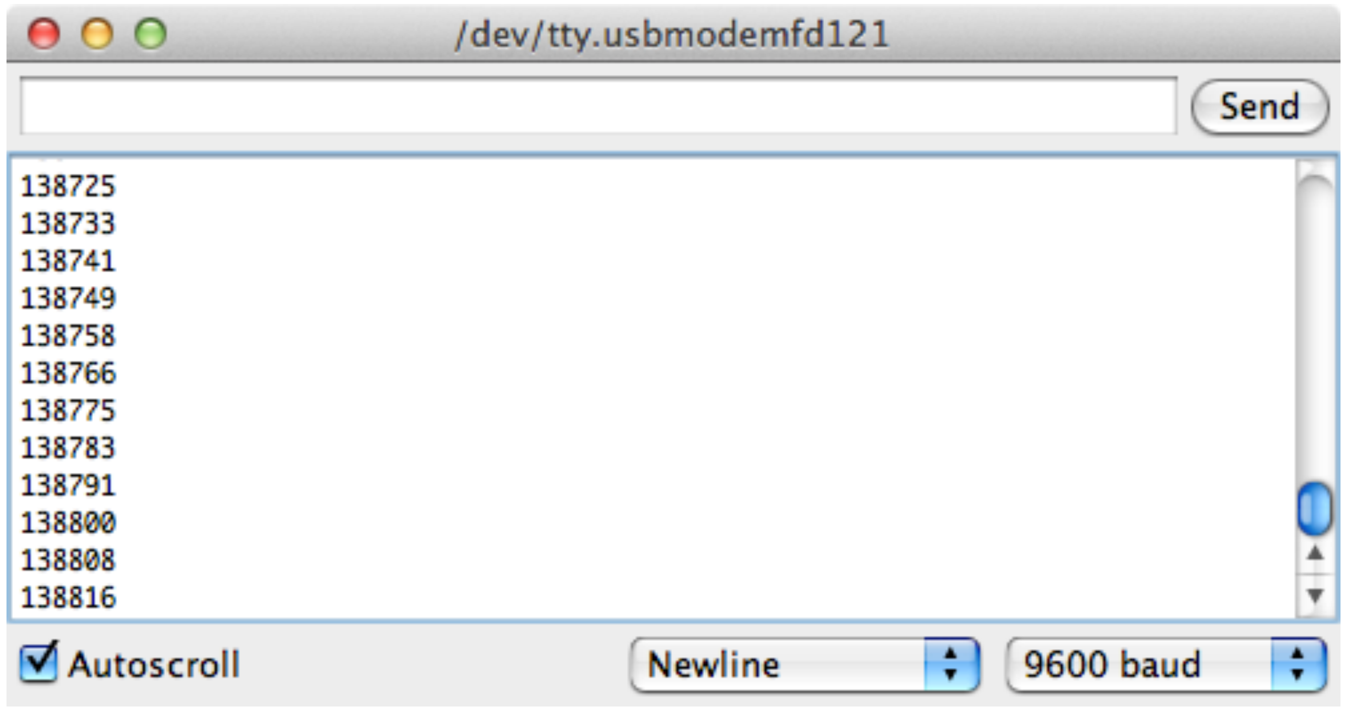
```
Done Saving.
```

14
Arduino Uno on /dev/tty.usbmodemfa131

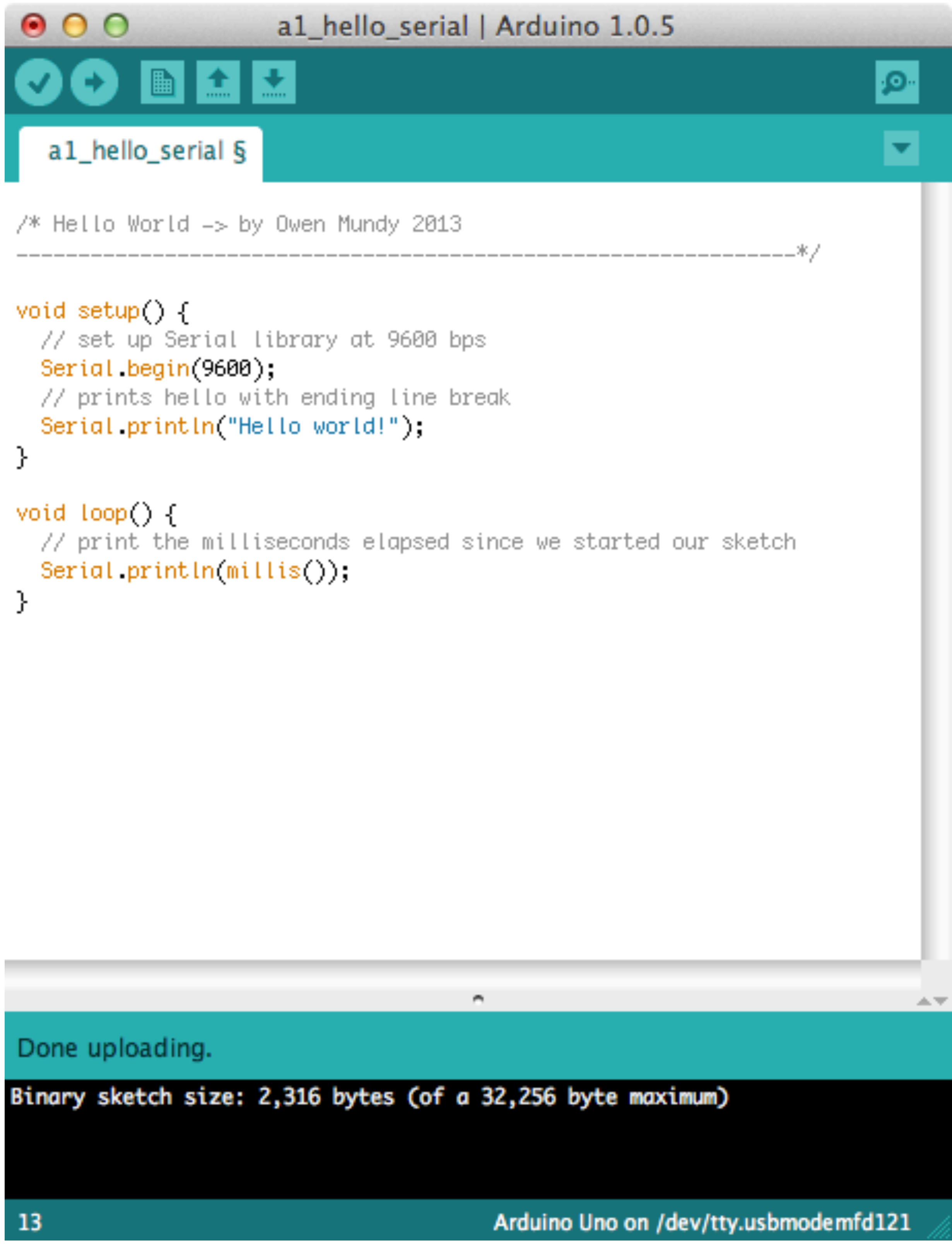
Serial communication

- We can put serial messages anywhere. This is helpful for keeping track of what’s going on behind the scenes.
- For example we could put the following code in the loop() function to see the number of milliseconds elapsed since we started our sketch.

Serial.println(millis());



```
138725
138733
138741
138749
138758
138766
138775
138783
138791
138800
138808
138816
```



a1_hello_serial | Arduino 1.0.5
a1_hello_serial §

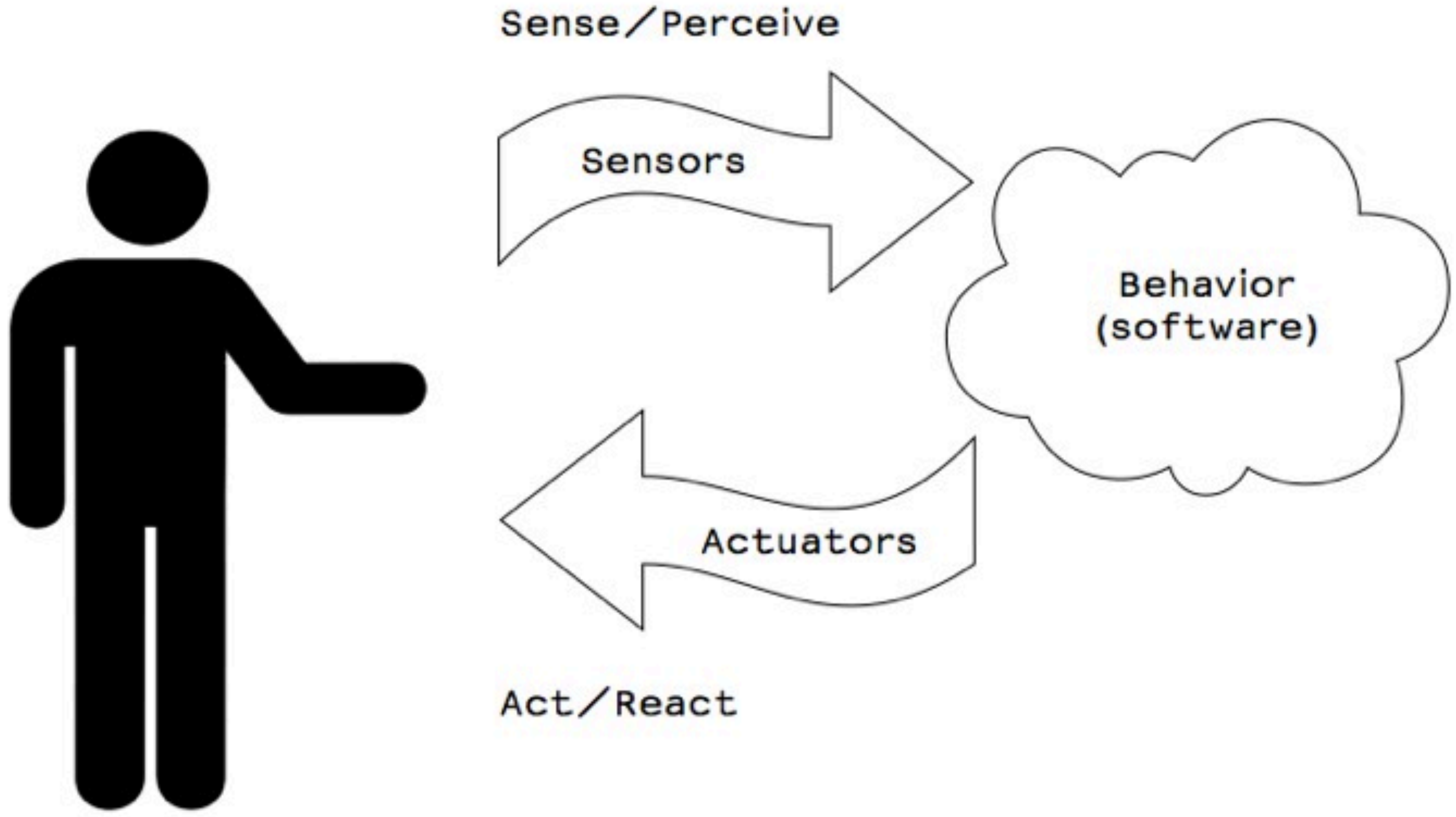
```
/* Hello World -> by Owen Mundy 2013
----------------------------------------------------------------*/

void setup() {
  // set up Serial library at 9600 bps
  Serial.begin(9600);
  // prints hello with ending line break
  Serial.println("Hello world!");
}

void loop() {
  // print the milliseconds elapsed since we started our sketch
  Serial.println(millis());
}
```

```
Done uploading.
Binary sketch size: 2,316 bytes (of a 32,256 byte maximum)
```

13
Arduino Uno on /dev/tty.usbmodemfd121

Sensors + actuators

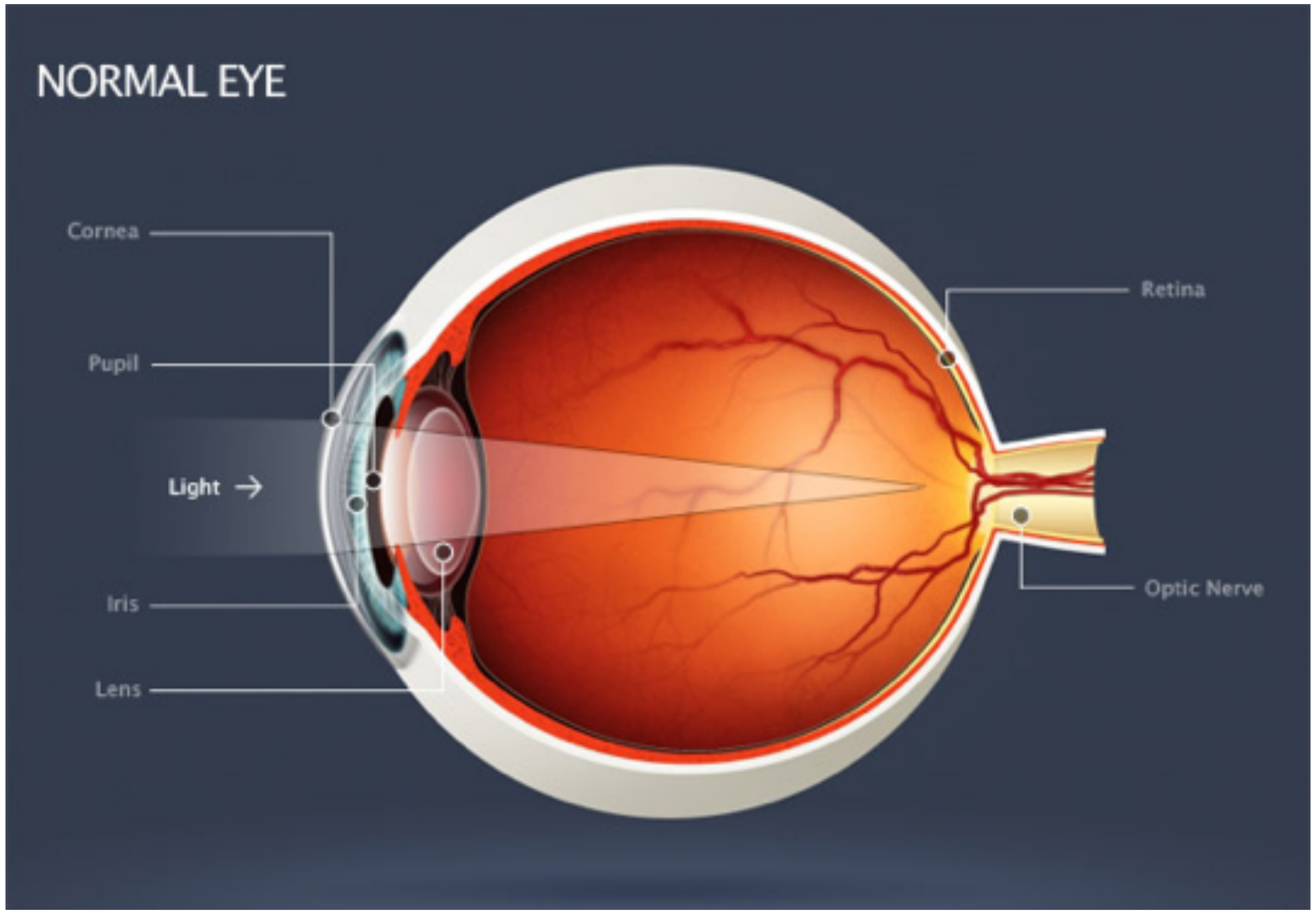
- In this lecture we'll learn to make the Arduino device respond to the real world.
- The devices will use sensors to measure some aspect of their environment and convert that information into electrical signals.
- We'll write software to process the sensor information and display the data in real time on our computers, as well as perform the data via some actuator.



Source: Getting Started with Arduino

Theory of operation

- The microcontroller is basically a very simple computer. But it can only process electric signals. *For example: Think of the electric pulses sent between neurons in our brains.*
- So to sense light, temperature, or other physical quantities, we first have to convert information about them into electricity. *Likewise, in our body, the eye converts light into signals that get sent to the brain using nerves.*
- For light, specifically, we can use a simple device called a *light dependent resistor* (a.k.a. *LDR*, or *photoresistor*) that can measure the amount of light that hits it and return it in the form electric signals to the microcontroller.



Source: Getting Started with Arduino

Theory of operation

- Once the electricity from the sensor has been read, the microcontroller, depending on conditions in our program, will act on the information, sending electrical signals back to an actuator (LED, motors, etc.). *In our bodies, for example, muscles receive electric signals from the brain and convert them into movement.*

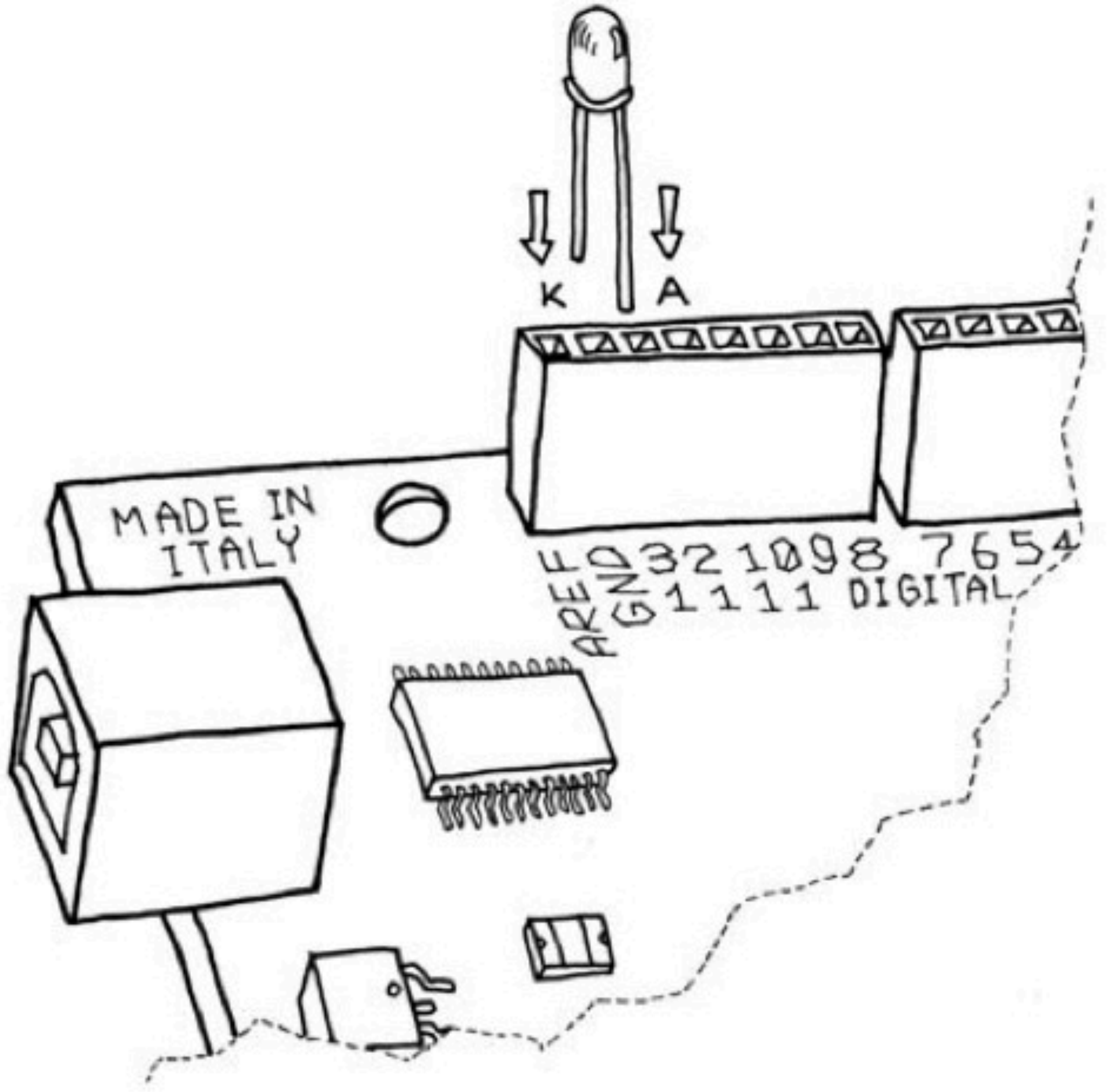
Source: Getting Started with Arduino

First, for testing: The digital i/o #13+LED

While Arduino boards have built-in LEDs, they also provide a quick way to test LEDs, and an even more obvious way to display signals read by the device. We can plug a through-hole LED directly into pin 13, which has a resistor built-in so that you won't burn-out the LED.

We can just change our previous sketch (blink.pde) to use this pin instead:

```
int ledpin = 13;
```



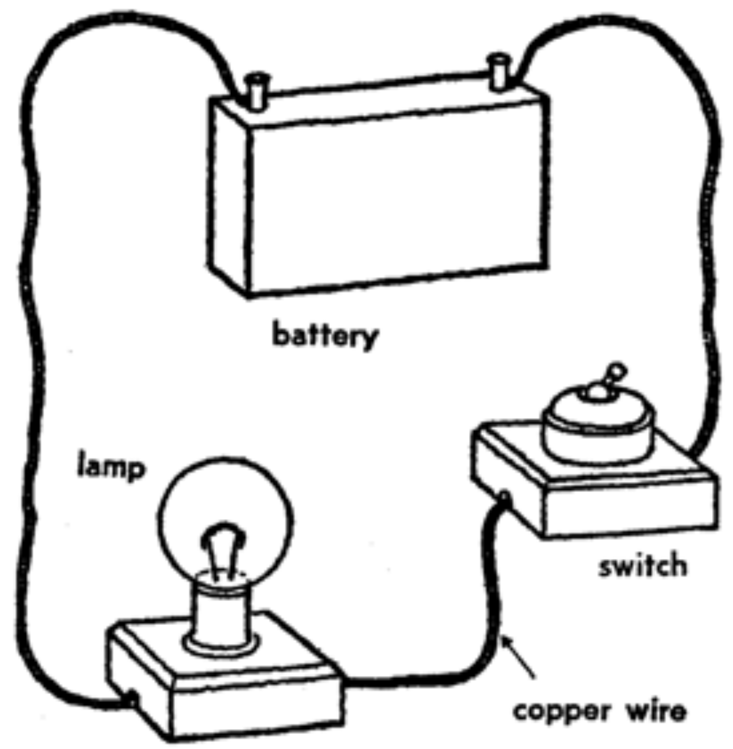
Source: Getting Started with Arduino

Switches -o⁄ o-

OK, so blinking an LED is easy, but it is only the beginning of what we can do with Arduino. In this previous example, the LED was our actuator, and our Arduino was controlling it. What we need now is a sensor to control our circuit.

We're going to start with the simplest form of sensor available: a ***pushbutton***.

If you take a pushbutton switch apart you would see that it is a very simple device: there are two bits of metal kept apart by a spring, and a plastic cap that when pressed brings the two bits of metal into contact.



When the bits of metal are apart, the circuit is *broken* or *open*, meaning current cannot circulate through our circuit. When we press it, we make a connection and allow current to flow. *A good example is a button on a water fountain.*

Source: Getting Started with Arduino

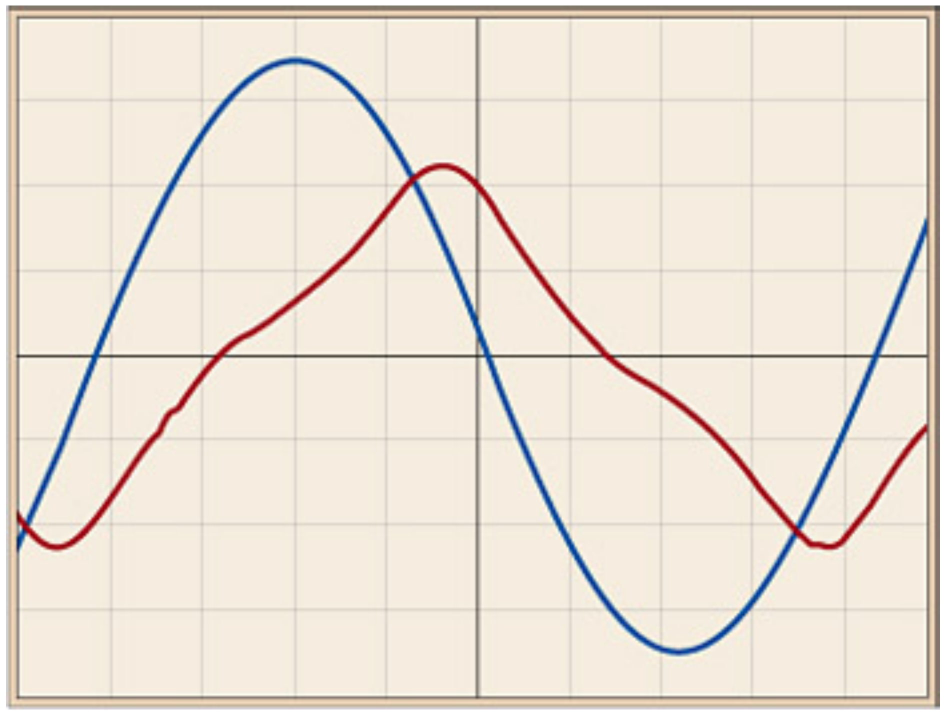
Switches

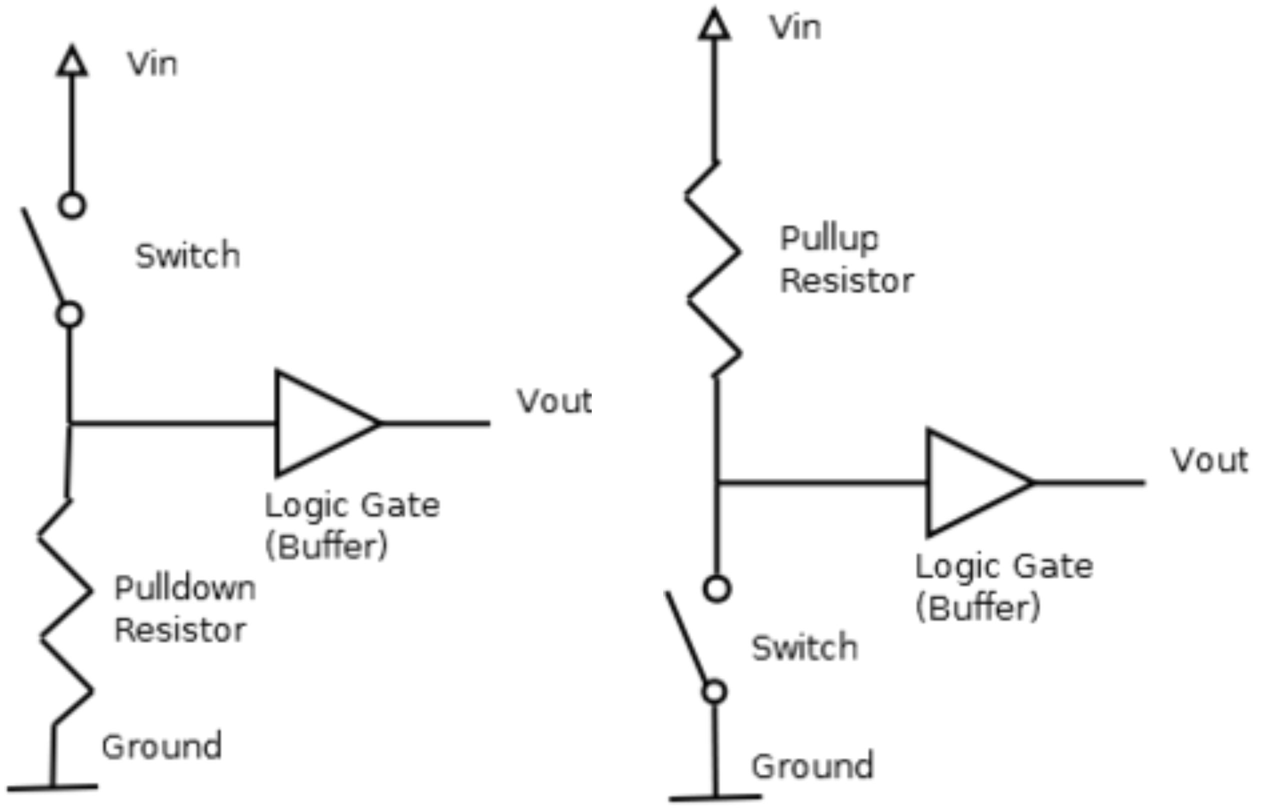
Before we make a circuit to read our switch, we should briefly discuss ***electrical noise***.

Reading digital signals can sometimes be tricky. When a circuit is open (button is not pressed) the current can fluctuate and we might inadvertently read that the button is pressed when it is not.

A ***pull-down resistor*** is a method for eliminating that noise, ensuring that there is a default value—the current allowed to flow past the resistor.

We place ***pull-down resistors*** between a switch or sensor and our ground, while ***pull-up resistors*** go between the 5V and the switch.



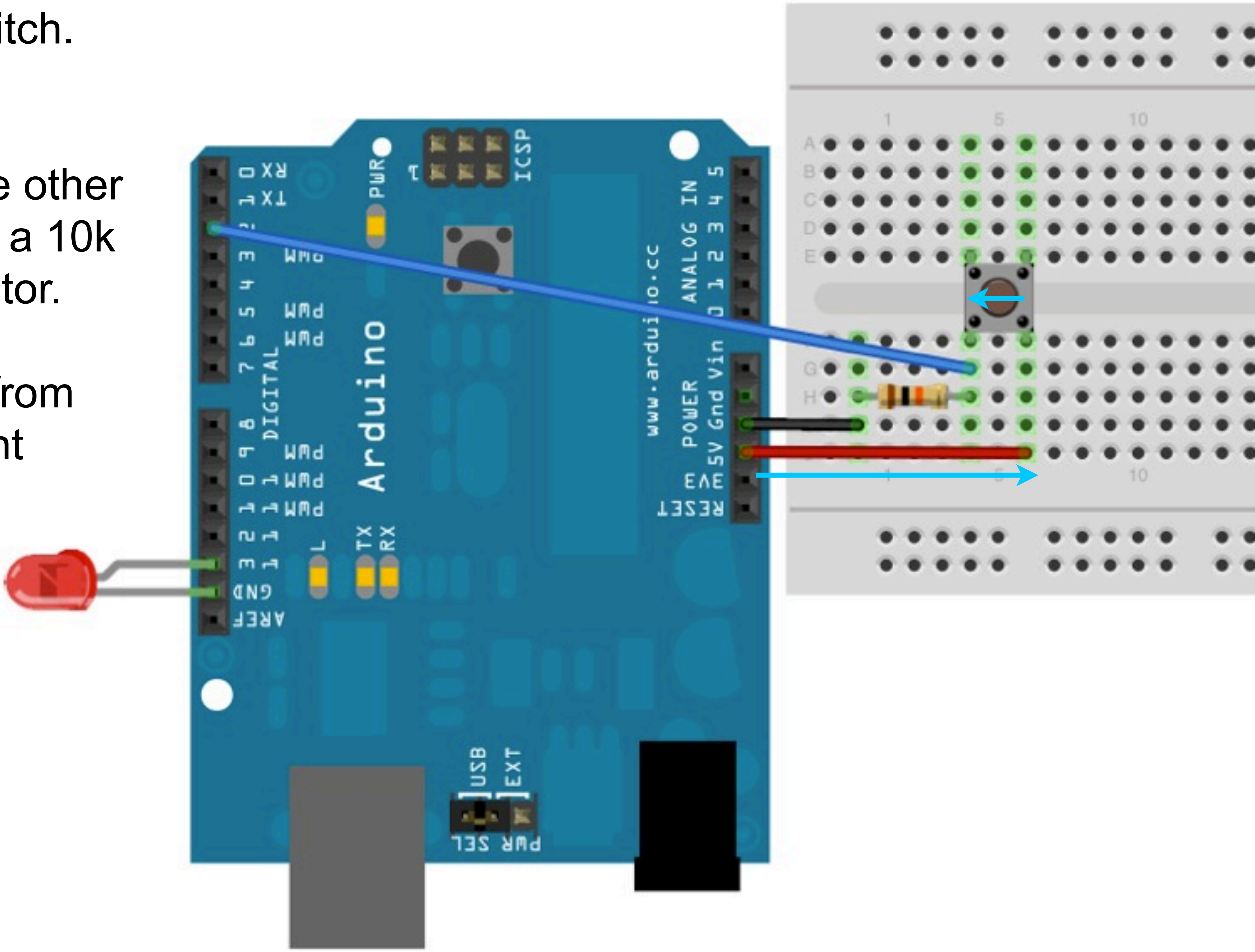


Switches

On our Arduino, we connect the **positive +** wire to one side of the switch.

Then we connect the **negative -** wire to the other side of the switch via a 10k ohm ***pull-down*** resistor.

Then we add a wire from digital i/o port #2 right before the resistor.



Switches

To monitor the state of a switch, we can use the ***digitalRead()*** function.

digitalRead() checks to see if there is voltage applied to a pin you specify between parentheses, and returns a value of HIGH or LOW, depending on its findings.

For example:

```
buttonState = digitalRead(buttonPin);
```

The other instructions that we've used so far haven't returned any information—they just executed what we asked them to do.

With digitalRead(), the Arduino can now talk to us, sending us information that we can use to make decisions immediately or later.

Switches

This code is much like our blink example. We start by introducing some variables that we will use in our program. Making one we’ll use to output signals to our LED (***ledPin***), another to keep track of signals from the pushbutton (***inPin***), and one to keep track of the pin status (***val***).

```
int ledPin = 13;
int inPin = 2;
int val = 0;
```

Then we write the setup() function, declaring the ***ledPin*** as output and the ***inPin*** as input:

```
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(inPin, INPUT);
}
```

Switches

And finally we add the loop function which will continuously read the input value and act on that input.

We use ***digitalRead()*** to read the input value from the button (***val***). If it is HIGH (button released) then we set the value of the ***ledPin*** to LOW (off), else if val ***is*** LOW, then we set the ***ledPin*** to HIGH.

```
void loop(){
  val = digitalRead(inPin);
  if (val == HIGH) {
    digitalWrite(ledPin, LOW);
  } else {
    digitalWrite(ledPin, HIGH);
  }
}
```

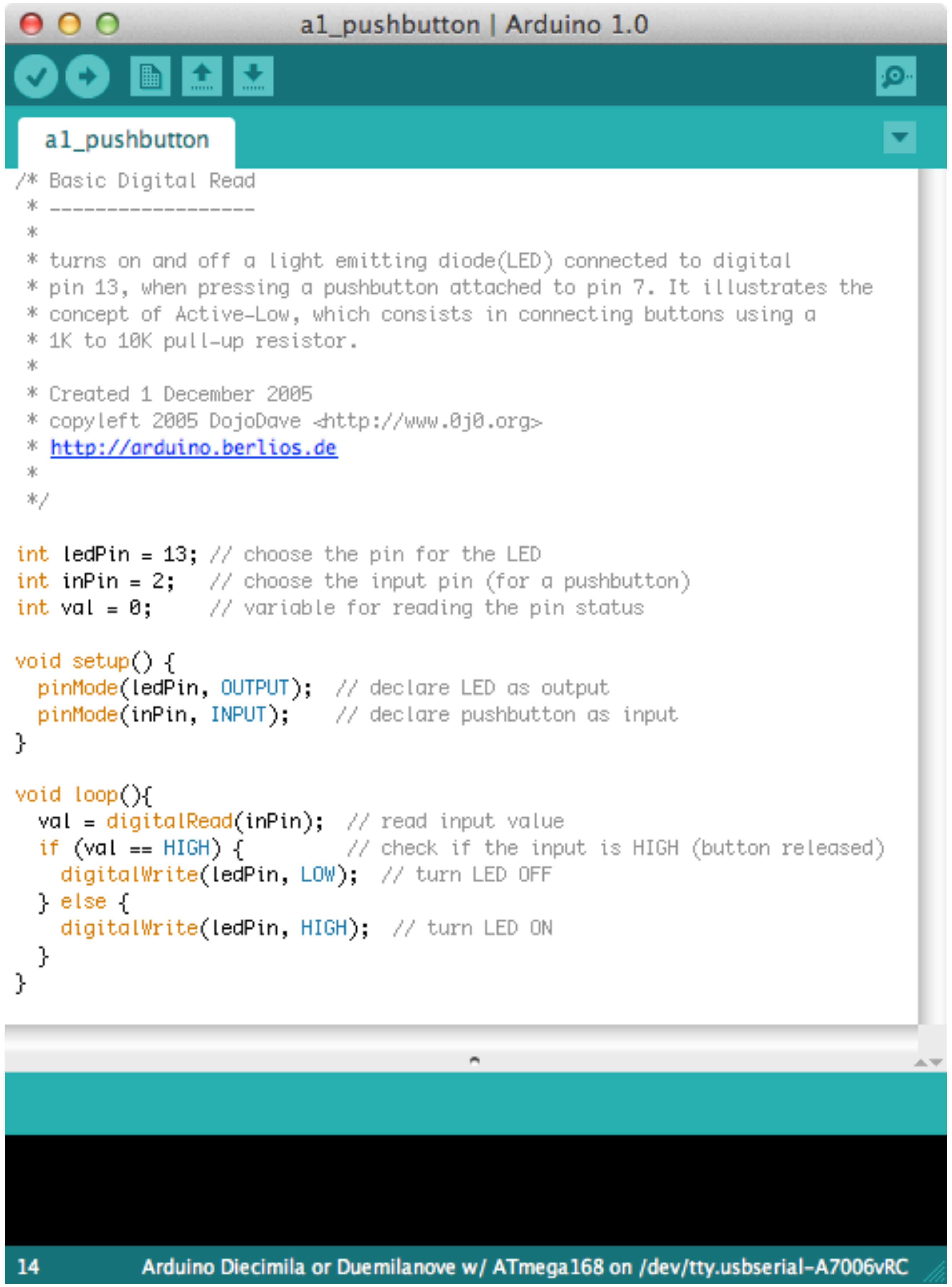
Switches

Here’s a summary of our code:

Now let’s play with this code. How to:

Make the Arduino count your button presses and display the total in the Console?

Advanced: Can you make it count your presses and blink the number of times back to you?

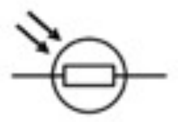


```
/* Basic Digital Read
 * ------------------
 *
 * turns on and off a light emitting diode(LED) connected to digital
 * pin 13, when pressing a pushbutton attached to pin 7. It illustrates the
 * concept of Active-Low, which consists in connecting buttons using a
 * 1K to 10K pull-up resistor.
 *
 * Created 1 December 2005
 * copyleft 2005 DojoDave <http://www.0j0.org>
 * http://arduino.berlios.de
 *
 */

int ledPin = 13; // choose the pin for the LED
int inPin = 2;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT);  // declare LED as output
  pinMode(inPin, INPUT);    // declare pushbutton as input
}

void loop(){
  val = digitalRead(inPin);  // read input value
  if (val == HIGH) {         // check if the input is HIGH (button released)
    digitalWrite(ledPin, LOW);  // turn LED OFF
  } else {
    digitalWrite(ledPin, HIGH);  // turn LED ON
  }
}
```

14 Arduino Diecimila or Duemilanove w/ ATmega168 on /dev/tty.usbserial-A7006vRC

Source: Arduino.cc: Pushbutton

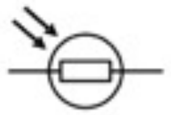
Light dependent resistor (LDR)



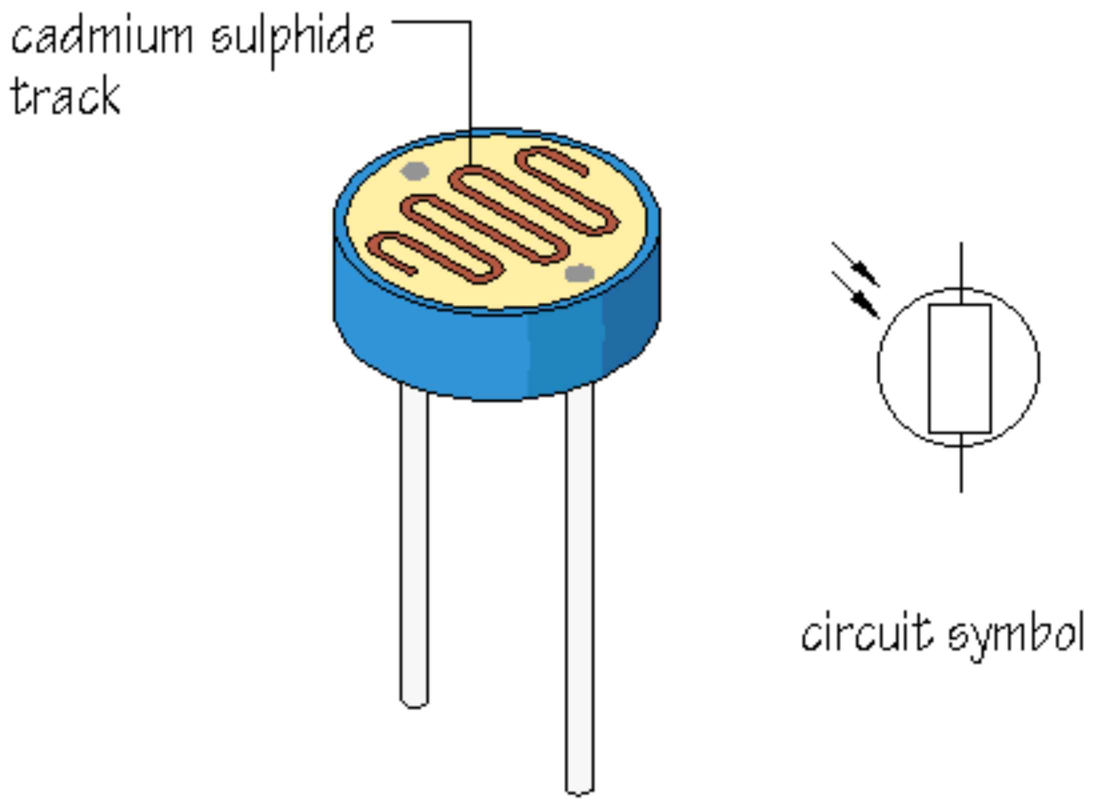
A **photoresistor** or **light dependent resistor** (**LDR**) is a type of sensor that can be use to measure the amount of light in an area. They are small, cheap, easy to use, and don't wear out.

At the heart of the LDR is a resistor whose resistance decreases with increasing incident light intensity. Meaning, in bright light, there is less resistance and more current is allowed to flow. In the dark and at low light levels, the resistance of an LDR is high, and little current can flow.

Light dependent resistor (LDR)

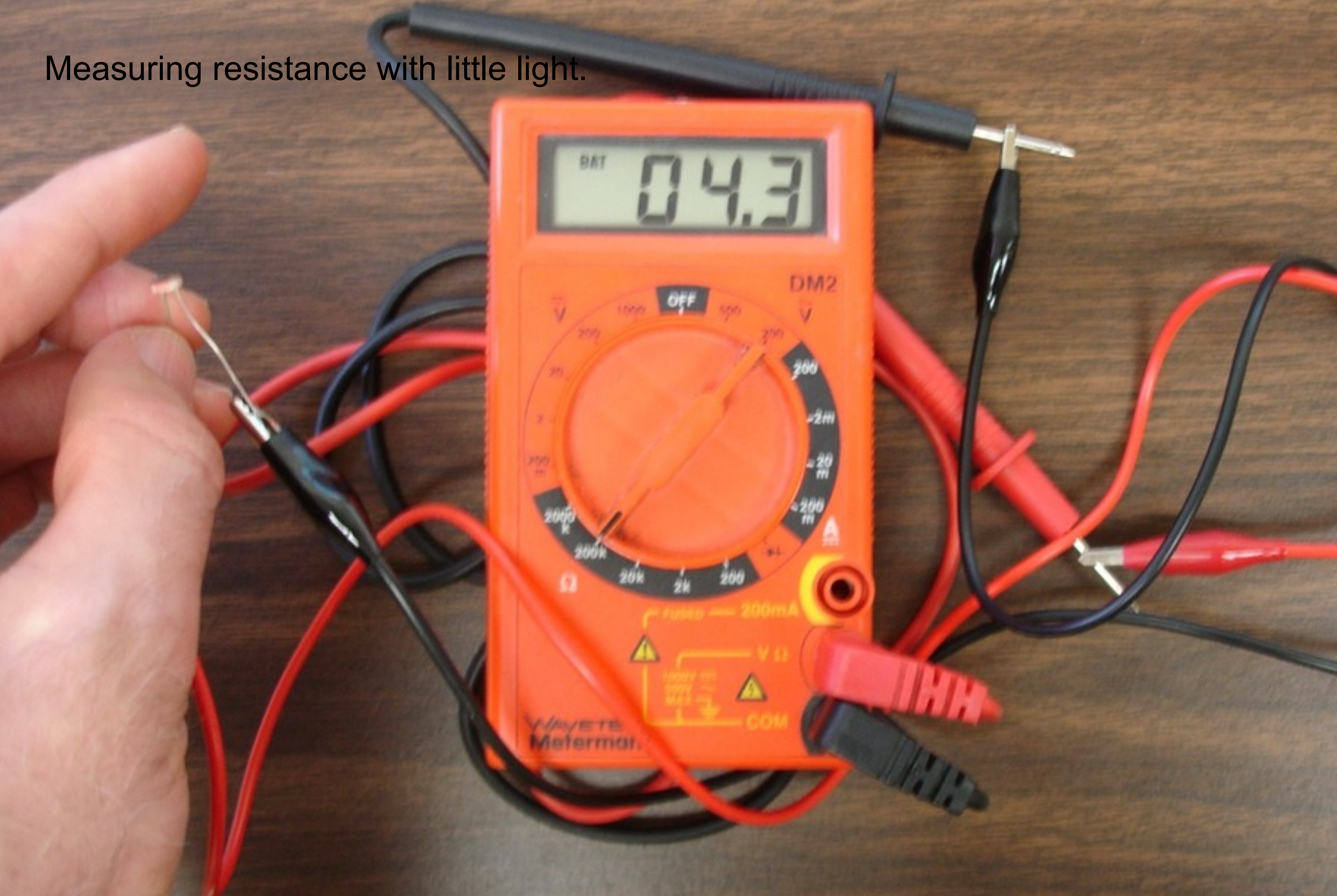


Examples:
Automatic security lights.
A solar powered flashlight.
Exposure control for a digital camera.
What else?



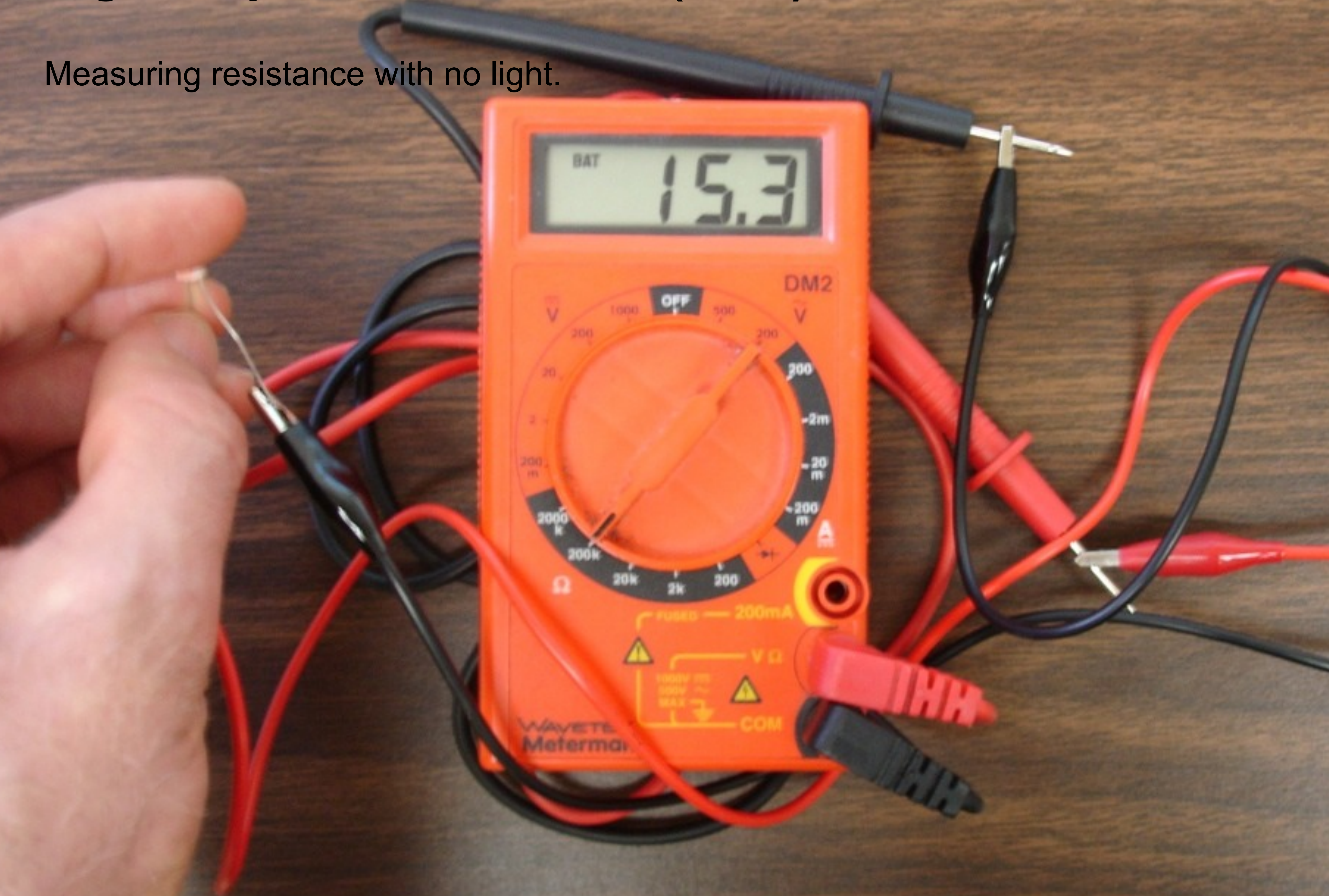
Light dependent resistor (LDR)

Measuring resistance with little light.

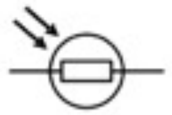


Light dependent resistor (LDR)

Measuring resistance with no light.

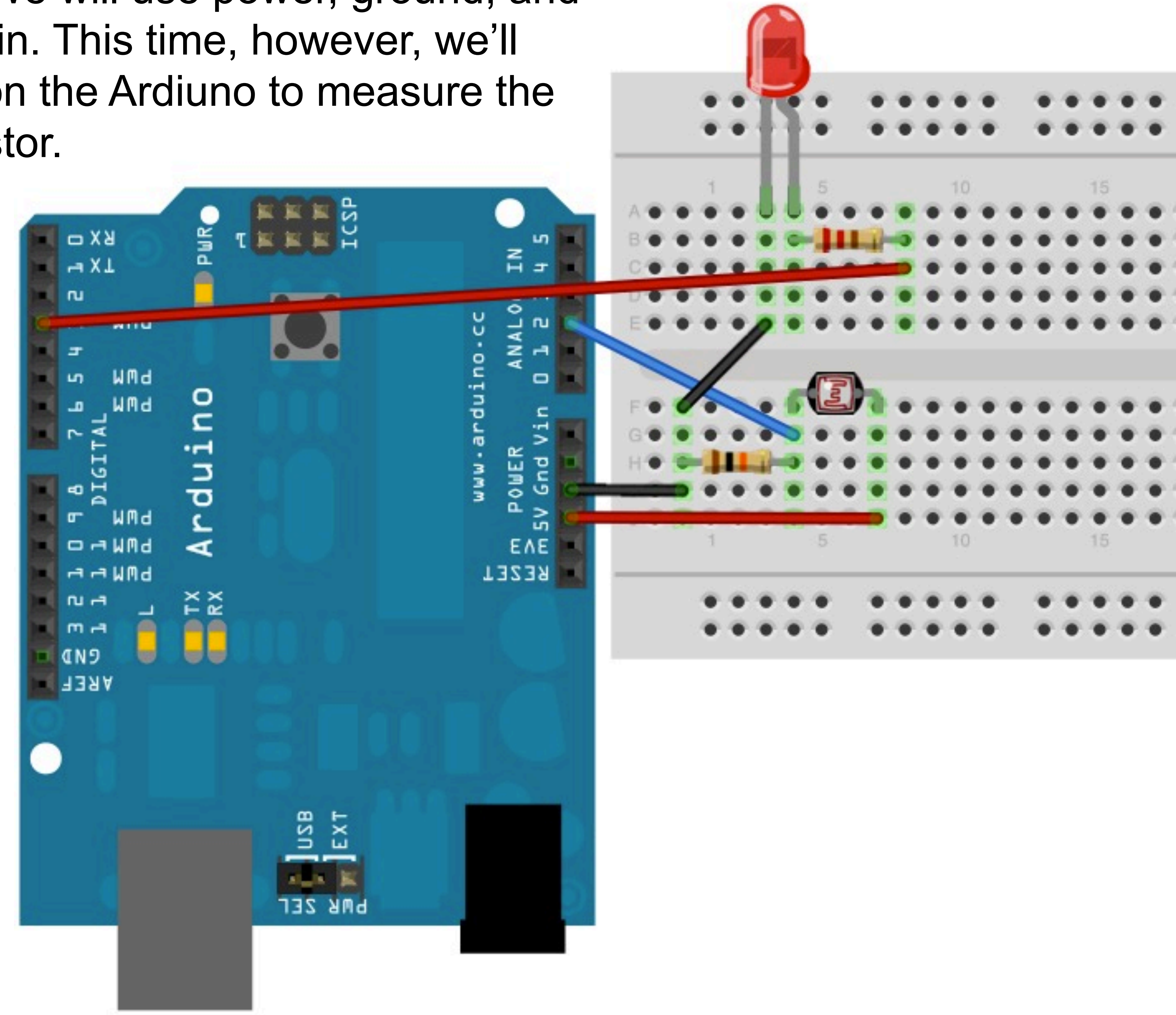


Light dependent resistor (LDR)

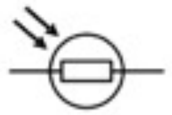


Wiring is pretty simple. We will use power, ground, and a pull-down resistor again. This time, however, we'll use an analog pin (#2) on the Ardiuno to measure the data from the photoresistor.

Also, add an LED and resistor so we can display the results.



Light dependent resistor (LDR)



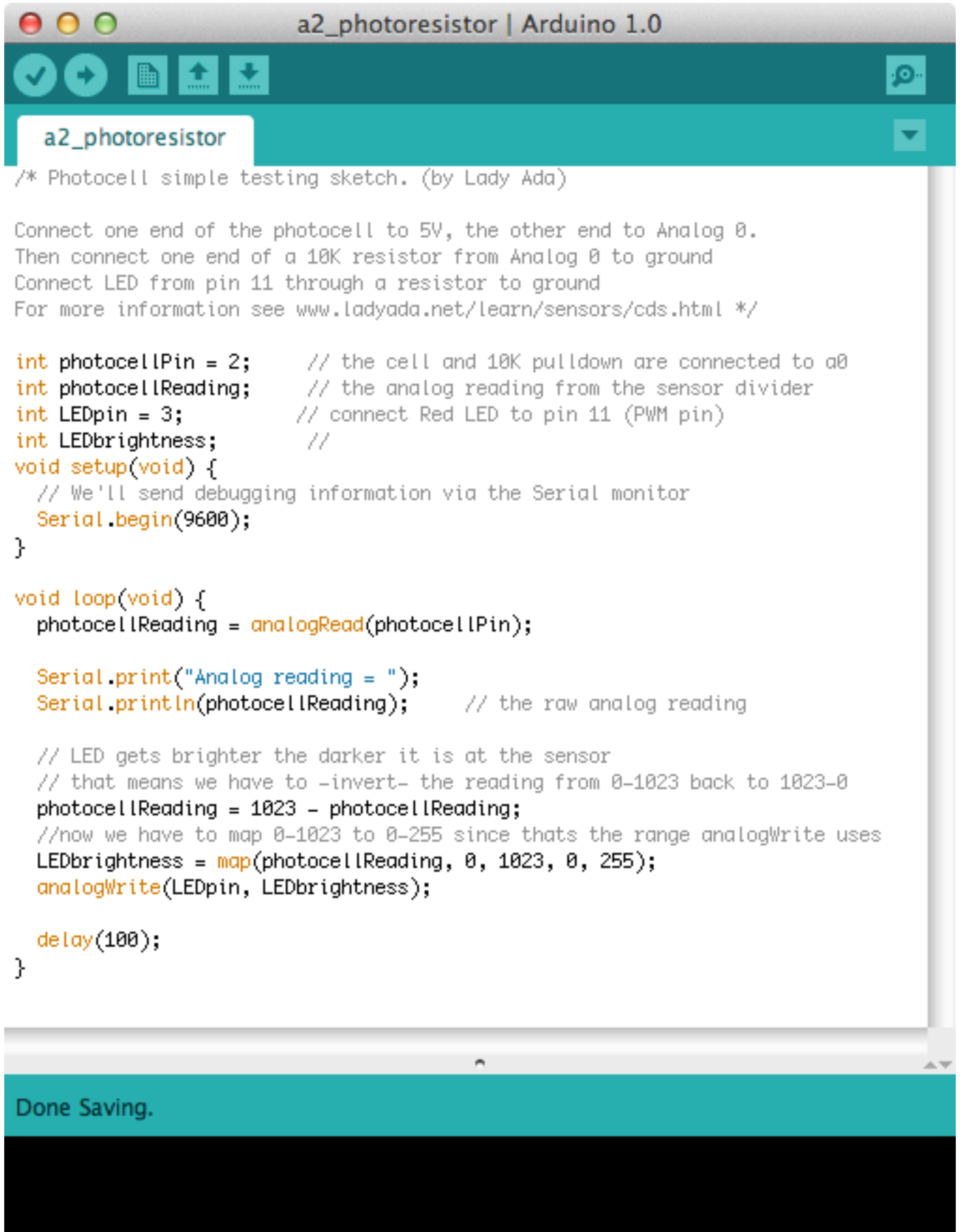
Here's what the code looks like.

1. We introduce four variables first.

2. In setup() we start a serial communication to get data back from the Arduino.

3. In loop() we use ***analogRead()*** to get the value from our LDR and print it with ***Serial.print()***.

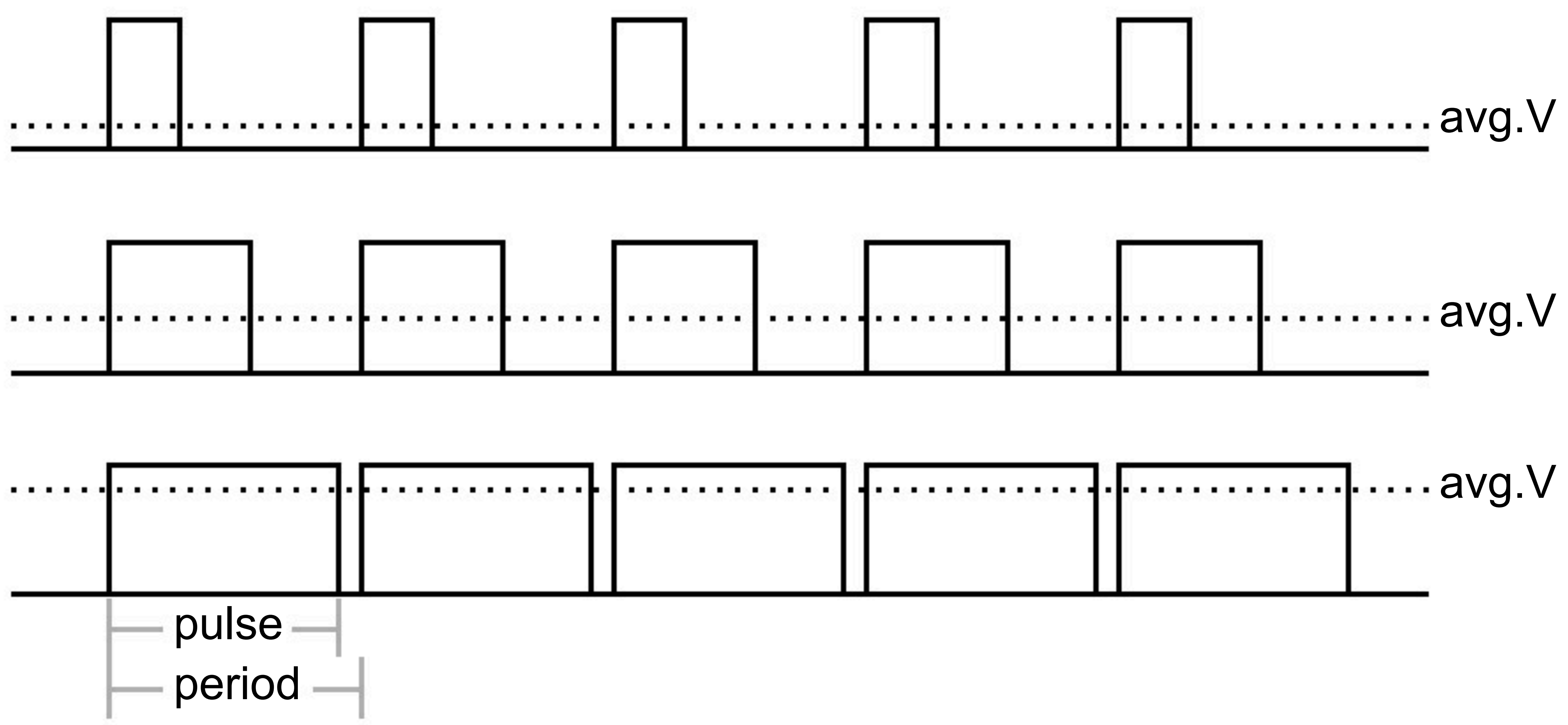
Then we use ***analogWrite()*** to write values to pin #3 (a PWM pin).



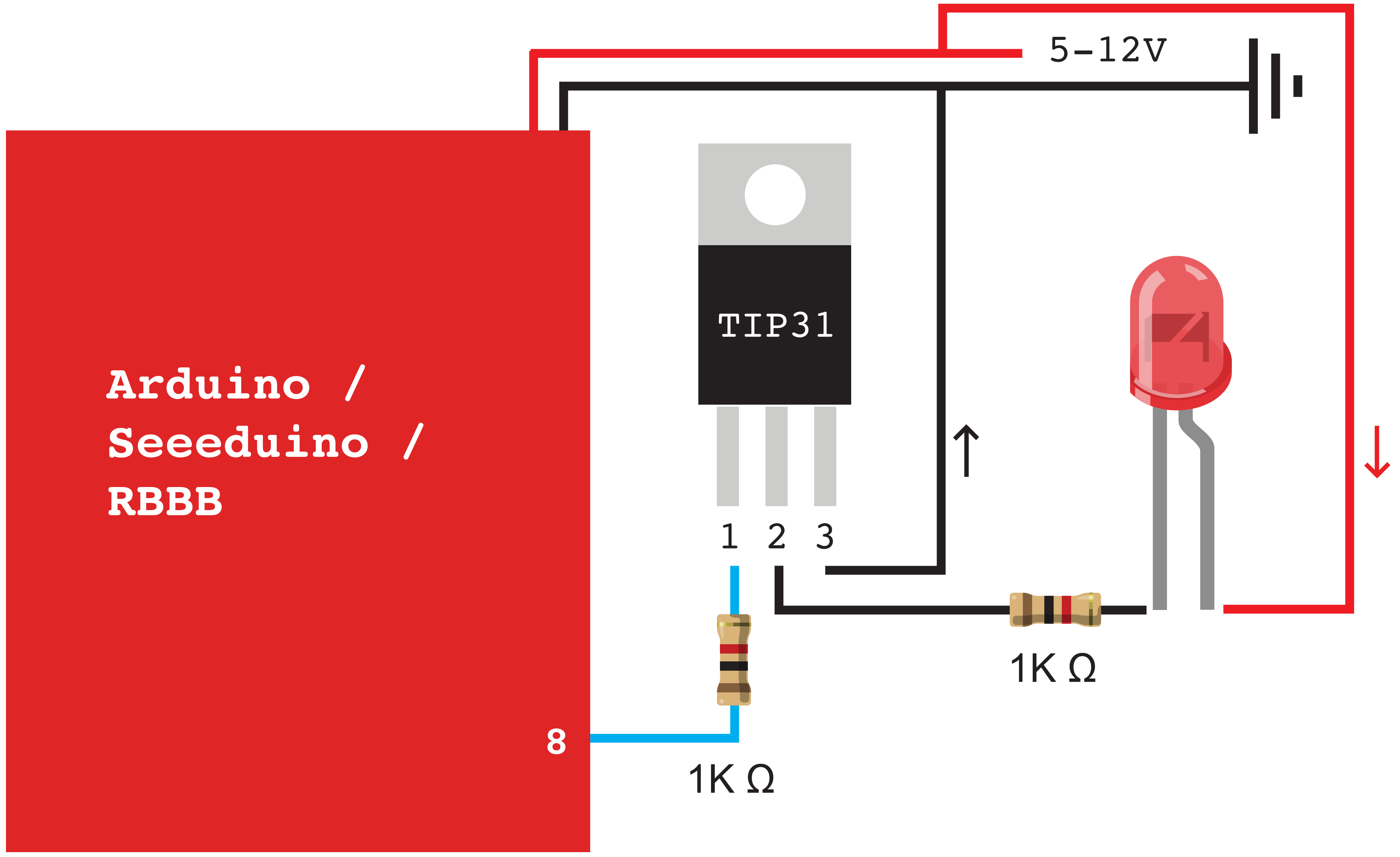
```
/* Photocell simple testing sketch. (by Lady Ada)

Connect one end of the photocell to 5V, the other end to Analog 0.
Then connect one end of a 10K resistor from Analog 0 to ground
Connect LED from pin 11 through a resistor to ground
For more information see www.ladyada.net/learn/sensors/cds.html */

int photocellPin = 2;     // the cell and 10K pulldown are connected to a0
int photocellReading;     // the analog reading from the sensor divider
int LEDpin = 3;          // connect Red LED to pin 11 (PWM pin)
int LEDbrightness;        //
void setup(void) {
  // We'll send debugging information via the Serial monitor
  Serial.begin(9600);
}

void loop(void) {
  photocellReading = analogRead(photocellPin);

  Serial.print("Analog reading = ");
  Serial.println(photocellReading);     // the raw analog reading

  // LED gets brighter the darker it is at the sensor
  // that means we have to -invert- the reading from 0-1023 back to 1023-0
  photocellReading = 1023 - photocellReading;
  //now we have to map 0-1023 to 0-255 since thats the range analogWrite uses
  LEDbrightness = map(photocellReading, 0, 1023, 0, 255);
  analogWrite(LEDpin, LEDbrightness);

  delay(100);
}
```

Pulse width modulation (PWM)

A final note regarding PWM, a form of pulsing electricity to achieve a desired Voltage level. The below graphic shows the average Voltage based on three sample pulse lengths. PWM is great for fading LEDs and controlling DC motors.



TIP31 used as a switch



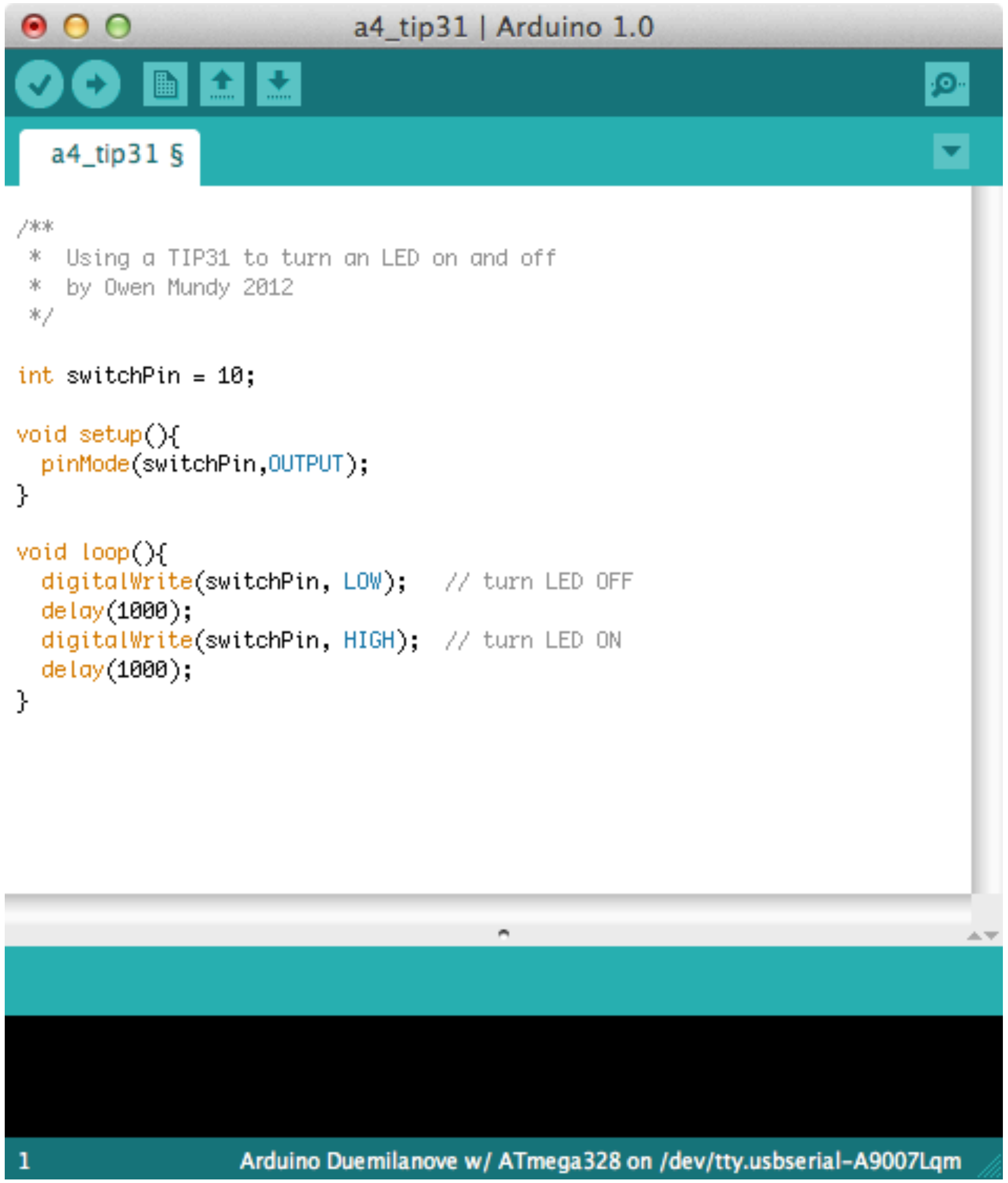
TIP31 AG|C Transistor used as a switch

• The Arduino, Tip31, and LED(s) must share a common ground.

“By putting a small voltage and current on the Base (1) you allow a larger current to flow from the Collector (2) to the Emitter (3).”
- p.99 of Physical Computing by Tom Igoe

TIP31 used as a switch

Here's what the code looks like.



```
/**
 *  Using a TIP31 to turn an LED on and off
 *  by Owen Mundy 2012
 */

int switchPin = 10;

void setup(){
  pinMode(switchPin,OUTPUT);
}

void loop(){
  digitalWrite(switchPin, LOW);   // turn LED OFF
  delay(1000);
  digitalWrite(switchPin, HIGH);  // turn LED ON
  delay(1000);
}
```

1   Arduino Duemilanove w/ ATmega328 on /dev/tty.usbserial-A9007Lqm