14. Tweak Arduino logo

Using serial communication, you'll use your Arduino to control a program on your computer.

When you program your Arduino, you're opening a connection between the computer and the microcontroller. You can use this connection to send data back and forth to other applications. The Arduino has a chip that converts the computer's USB-based communication to the serial communication the Arduino uses. Serial communication means that the two computers, your Arduino and PC, are exchanging bits of information serially, or one after another in time.

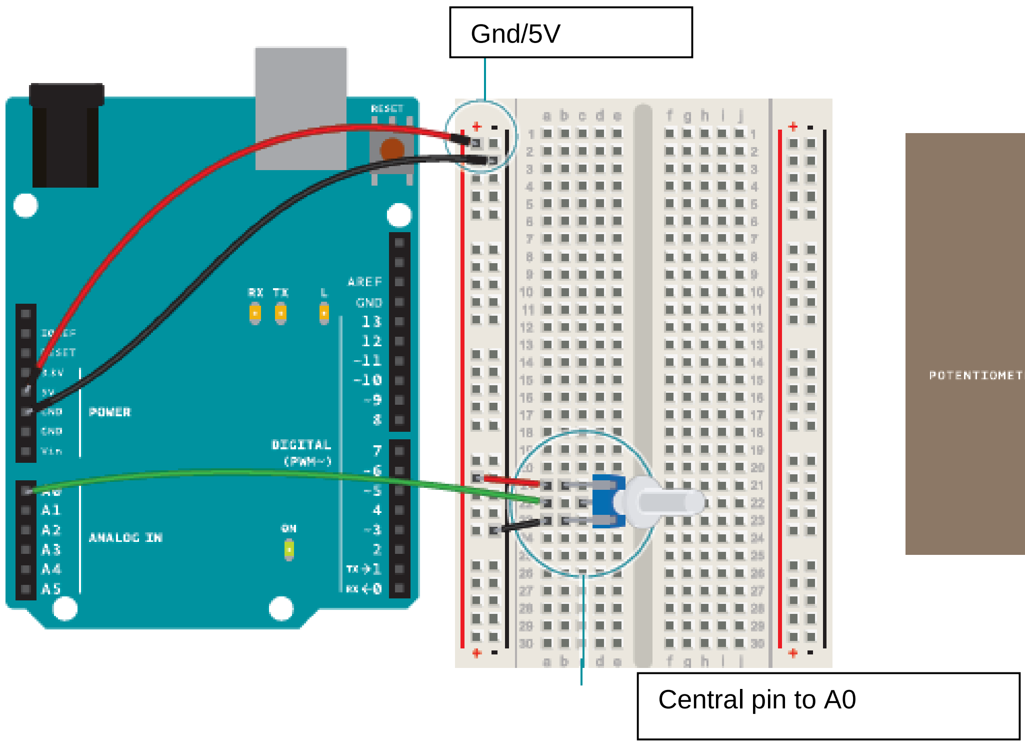
When communicating serially, computers need to agree on the speed at which they talk to one another. You've probably noticed when using the serial monitor there's a number at the bottom right corner of the window. That number, 9600 bits per second, or baud, is the same as the value you've declared using Serial.begin(). That's the speed at which the Arduino and computer exchange data.

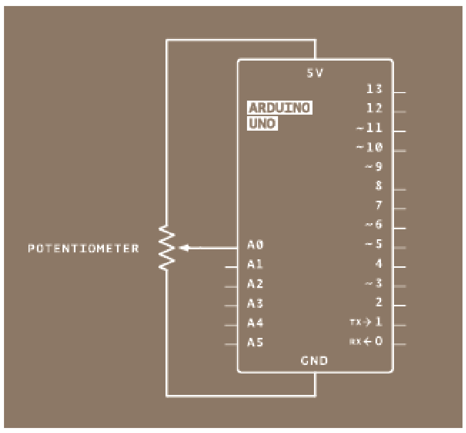
You've used the serial monitor to look at values from the analog inputs; you'll use a similar method to get values into a program you're going to write in a programming environment called Processing. Processing is based on Java, and Arduino's programming environment is based on Processing's. They look pretty similar, so you should feel right at home there.

The most efficient way to send data between the Arduino and Processing is by using the Serial.write() function in Arduino. It's similar to the Serial.print() function you've been using in that it sends information to an attached computer, but instead of sending human readable information like numbers and letters, it sends values between 0-255 as raw bytes. This limits the values that the Arduino can send, but allows for quick transmission of information.

On both your computer and Arduino, there's something called the serial buffer which holds onto information until it is read by a program. You'll be sending bytes from the Arduino to Processing's serial buffer. Processing will then read the bytes out of the buffer. As the program reads information from the buffer, it clears space for more.

When using serial communication between devices and programs, it's important that both sides not only know how fast they will be communicating, but also agree on what is sent and received.





```
void setup() {
  // initialize serial communication
  Serial.begin(9600);
}

void loop() {
  // read the value of A0, divide by 4 and
  // send it as a byte over the serial connection
  Serial.write(analogRead(A0) / 4);
  delay(1);
}
```

First, program your Arduino. In setup(), you'll start serial communication, just as you did earlier when looking at the values from an attached sensor. The Processing program you write will have the same serial baud rate as your Arduino.
In the loop(), you're going to use the Serial.write() command to send information over the serial connection.
Serial.write() can only send a value between 0 and 255. To make sure you're sending values that fit within that range, divide the analog reading by 4.
After sending the byte, wait for one millisecond to let the ADC settle down. Upload the program to the Arduino then set it aside while you write your Processing sketch.

```
void setup() {
  size(1, 1);
  surface.setResizable(true);
  // set the color mode to Hue/Saturation/Brightness
  colorMode(HSB, 255);

  // load the Arduino logo into the PImage instance
  //the picture should be in the same folder as Processing
  logo = loadImage("logo(170x116).png");

  // make the window the same size as the image
  surface.setSize(170, 116);

  // print a list of available serial ports to the
  // Processing staus window
  println("Available serial ports:");
  println(Serial.list());

  // Tell the serial object the information it needs to communicate
  // with the Arduno. Change Serial.list()[0] to the correct
  // port corresponding to your Arduino board.  The last
  // parameter (e.g. 9600) is the speed of the communication.  It
  // has to correspond to the value passed to Serial.begin() in your
  // Arduino sketch.
 myPort = new Serial(this, Serial.list()[0], 9600);
```

```
void draw() {

  // if there is information in the serial port
  if ( myPort.available() > 0) {
    // read the value and store it in a variable
    bgcolor = myPort.read();

    // print the value to the status window
    println(bgcolor);
  }

  // Draw the background. the variable bgcolor
  // contains the Hue, determined by the value
  // from the serial port
  background(bgcolor, 255, 255);

  // draw the Arduino logo
  image(logo, 0, 0);
}
```

The Processing language is similar to Arduino, but there are enough differences that you should look at some of their tutorials and the "Getting Started" guide to familiarize yourself with the language.
Open a new Processing sketch. Processing, unlike the Arduino, doesn't know about serial ports without including an external library. Import the serial library.
You need to create an instance of the serial object, just like you've done in Arduino with the Servo library. You'll use this uniquely named object whenever you want to use the serial connection.
To use images in Processing, you need to create an object that will hold the image and give it a name.

Create a variable that will hold the background hue of the Arduino logo. The logo is a .png file, and it has built-in transparency, so it's possible to see the background color change.
Processing has a setup() function, just like Arduino. Here's where you'll open the serial connection and give the program a couple of parameters that will be used while it runs.
You can change the way Processing works with color information. Typically, it works with colors in a Red Green Blue (RGB) fashion. This is similar to the color mixing you did in Project 4, when you used values between 0 and 255 to change the color of an RGB LED. In this program, you're going to use a color mode called HSB, which stands for Hue, Saturation, and Brightness. You'll change the hue when you turn the potentiometer.
colorMode() takes two arguments: the type of mode, and the maximum value it can expect.
To load the Arduino image into the sketch, read it into the logo object you created earlier.
With the size() function, you tell Processing how large the display window will be. If you use logo.width and logo.height as the arguments, the sketch will automatically scale to the size of the image you're using. Processing has the ability to print out status messages using the println() command. I
f you use this in conjunction with the Serial.list() function, you'll get a list of all the serial ports your computer has when the program first starts. You'll use this once you're finished programming to see what port your Arduino is on.
You need to tell Processing information about the serial connection. To populate your named serial object myPort with the necessary information, the program needs to know it is a new instance of the serial object. The parameters it expects are which application it will be speaking to, which serial port it will communicate over, and at what speed.
The draw() function is analogous to Arduino's loop() in that it happens over and over again forever. This is where things are drawn to the program's window.

Check if there is information from the Arduino. The myPort.available() command will tell you if there is something in the serial buffer. If there are bytes there, read the value into the bgcolor variable and print it to the debug window.
The function background() sets the color of the window. It takes three arguments. The first argument is the hue, the next is brightness, and the last is saturation. Use the variable bgcolor as the hue value, and set the brightness and saturation to the maximum value, 255.
You'll draw the logo with the command image(). You need to tell image() what to draw, and what coordinates to start drawing it in the window. 0,0 is the top left, so start there.