

# TWEAK THE ARDUINO LOGO

---

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

Discover : serial communication with a computer program, Processing

Time : 45 minutes

Level : ■■■■■■

Builds on projects : 1,2,3

*You've done a lot of cool stuff with the physical world, now it's time to control your computer with the Arduino. 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 a 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. A bit is the smallest amount of information a computer can understand.

You've used the serial monitor to look at the 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**. They look pretty similar, so you should feel right at home there.



Before getting started with the project, download the latest version of Processing from <http://www.processing.org>. It may be helpful to look at the "Getting started" and "Overview" tutorials at <http://www.processing.org/learning>. These will help you to familiarize yourself with Processing before you start writing software to communicate with your Arduino.

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 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 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 what they should be expecting. When you meet someone, you probably expect a "Hello!"; if instead they say something like "The cat is fuzzy", chances are you will be caught off guard. With software, you will need to get both sides agree on what is sent and received.

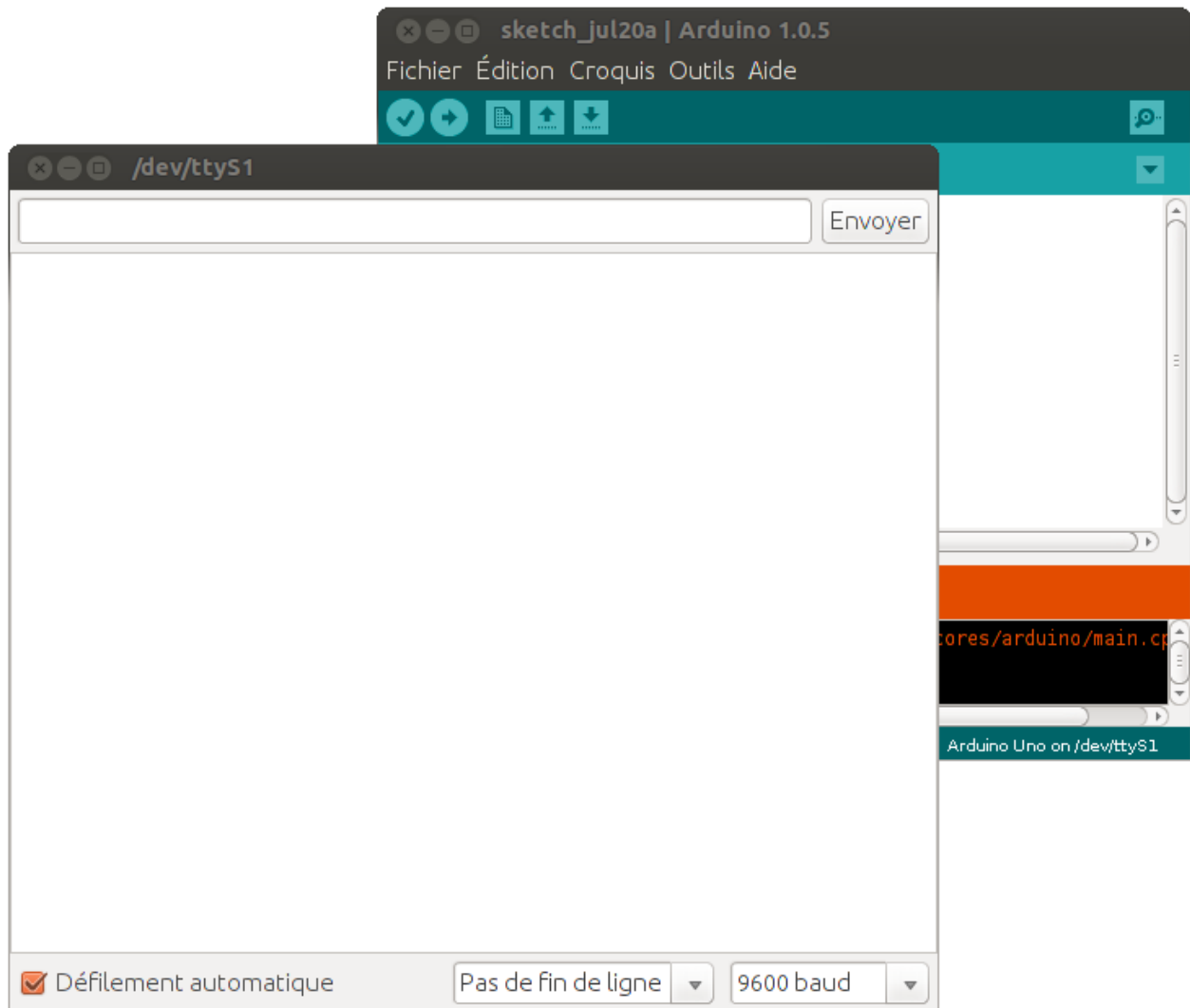
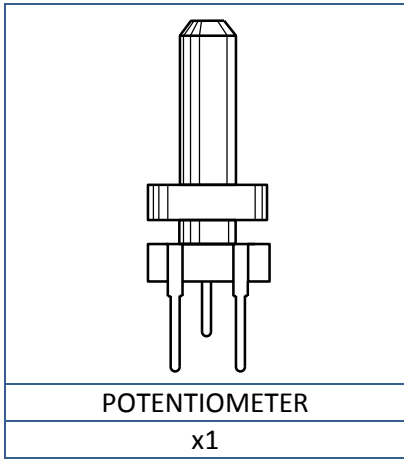


Fig.1

## INGREDIENTS



POTENTIOMETER

x1

## BUILD THE CIRCUIT

Fig.2 - [Circuit]

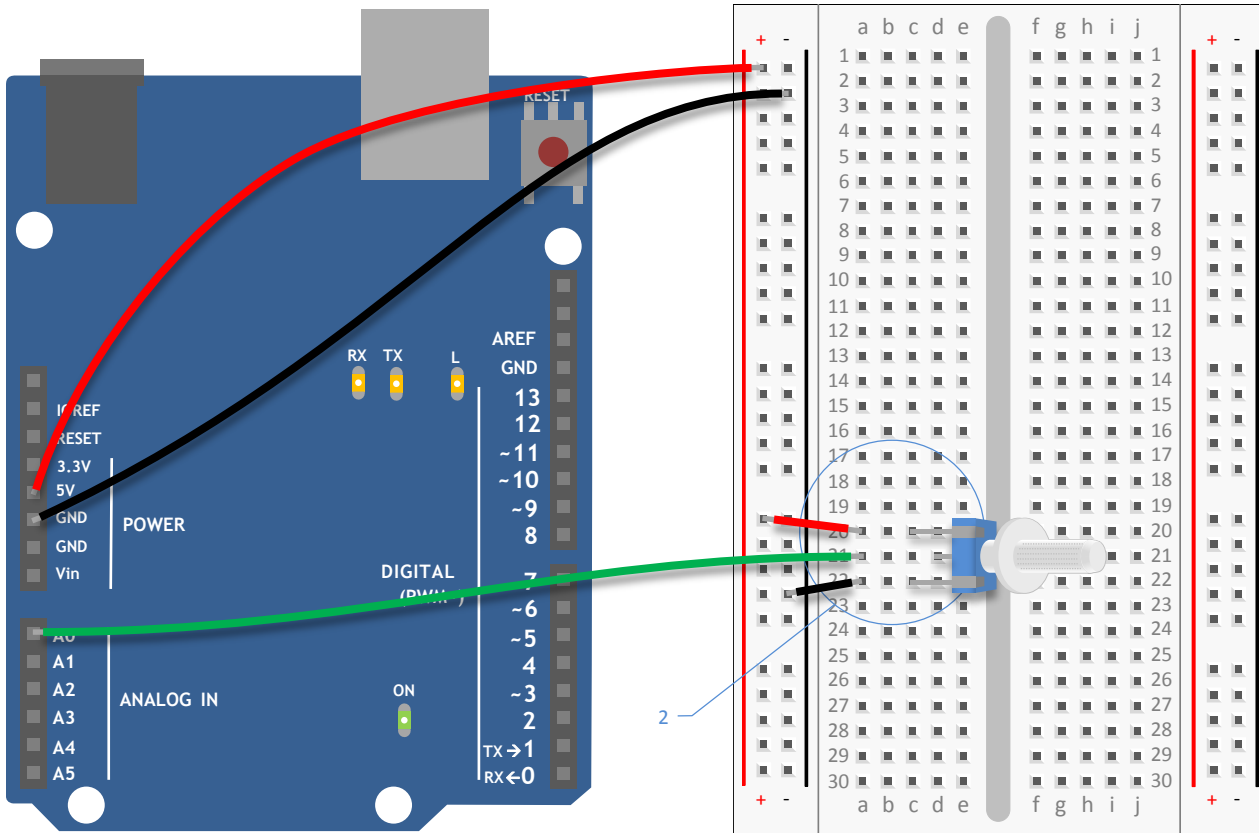
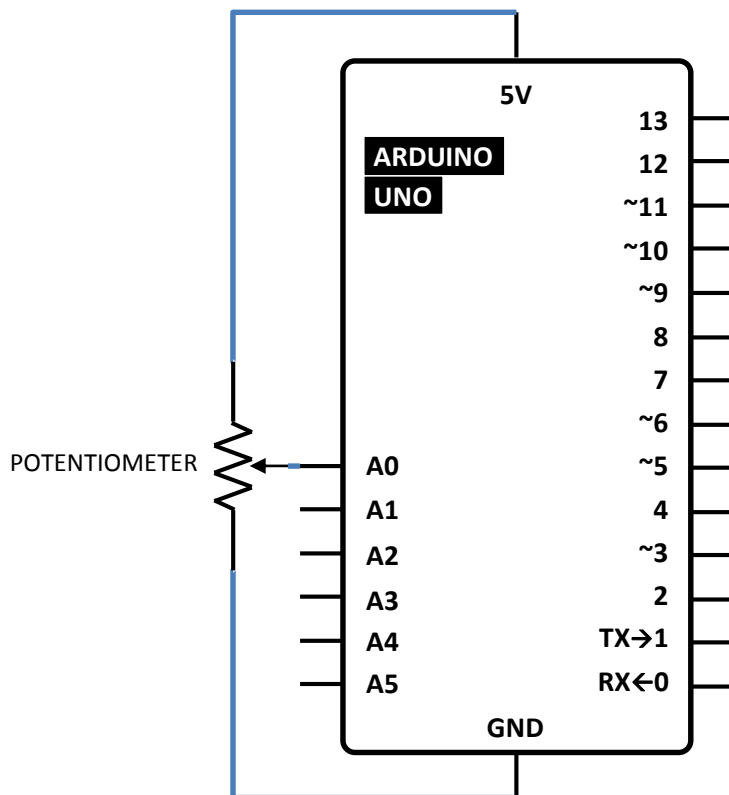


Fig.3 - [Schematic]



- 1) Connect power and ground to your breadboard.
- 2) Connect each end of your potentiometer to power and ground. Connect the middle leg to AnalogIn pin 0.

## THE ARDUINO CODE

### *Open a serial connection*

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.

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

### *Send the sensor value*

In the **loop()**, you're going to use the **Serial.write()** command to send information over the serial communication. **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.

```
void loop() {  
    Serial.write(analogRead(A0)/4);  
}
```

### *Let the ADC stabilize*

After sending the byte, wait for **100** milliseconds to let the ADC settle down.

```
    delay(100);  
}
```

Upload the program to the Arduino then set it aside while you write your Processing sketch.

**SAVE AND CLOSE THE ARDUINO IDE NOW, LET'S START THE PROCESSING IDE**

## THE PROCESSING CODE

### Import and set up the serial object

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 mentioned before 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.

```
import processing.serial.*;
Serial myPort;
```

### Create an object for the image

To use images in Processing, you need to create an object that will hold the image and give it a name.

```
PImage logo;
```

### Variable to store the background color

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.

```
int bgcolor = 0 ;
```

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.

```
void setup() {
```

### Setting the color mode

You can change the way Processing works with color information. Typically, it works with colors in 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.

```
colorMode(HSB, 255);
```

### Loading the image

To load the Arduino image into the sketch, read it into the logo object created earlier. When you supply the URL of an image, Processing will download it when you run the program.

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.

```
logo = loadImage("http://arduino.cc/logo.png");
size(logo.width, logo.height);
```

### Printing available serial ports

Processing has the ability to print out status messages using the **println()** command. If 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.

```
println("Available serial ports:");
printArray(Serial.list());
```

### Creating the serial object

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.

```
myPort = new Serial(this, Serial.list()[0], 9600);
}
```

The attribute **this** tells Processing you're going to use the serial connection in this specific application. The **Serial.list()[0]** argument specifies which serial port you're using. **Serial.list()** contains an array of all the attached serial devices. The argument **9600** should look familiar, it defines the speed at which the program will communicate.

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.

```
void draw() {
```

### Reading Arduino data from the serial port

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 values into the **bgcolor** variable and print it to the debug window.

```
if (myPort.available() > 0) {
    bgcolor = myPort.read();
    println(bgcolor);
}
```

### Setting the image background and displaying the image

The function **background()** sets the color of the window. It takes three arguments. The first argument is the hue, the next is the 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.

```
background(bgcolor, 255, 255);
image(logo, 0, 0);
}
```

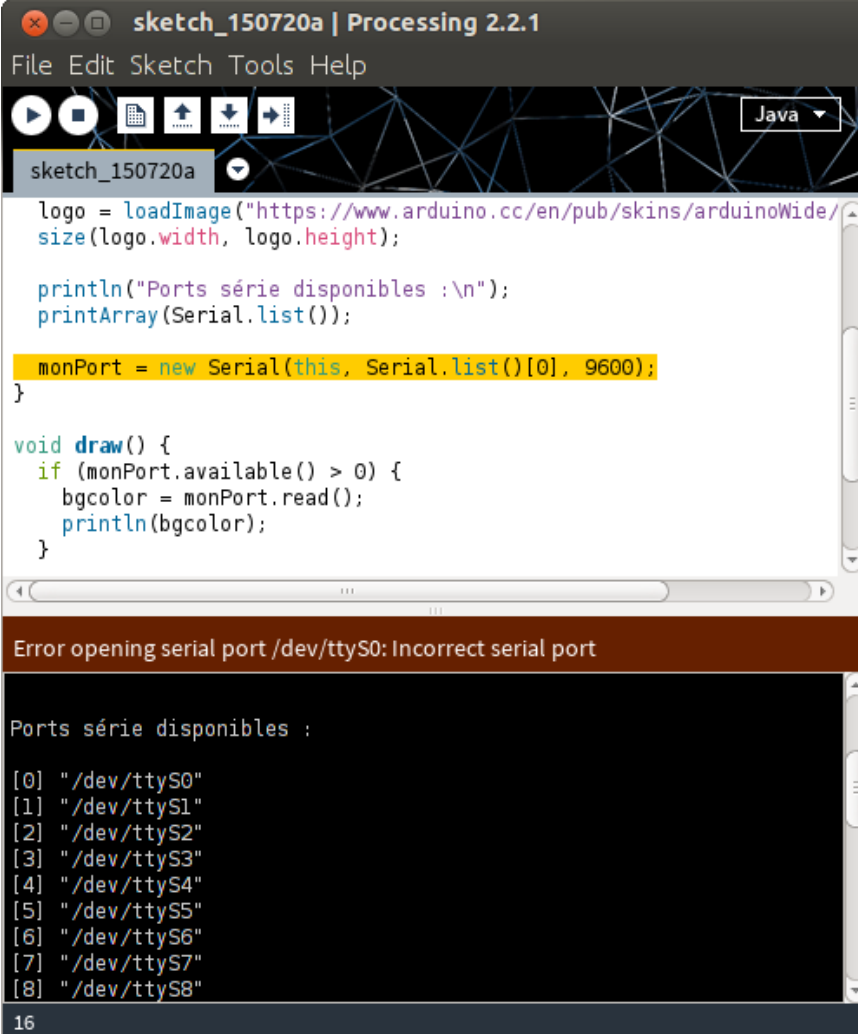


## USE IT

Connect your Arduino and open the serial monitor. Turn the pot on your breadboard. You should see a number of characters as you twist the knob. The serial monitor expects ASCII characters, not raw bytes. ASCII is information encoded to represent text in computers. What you see in the window is the serial monitor trying to interpret the bytes as ASCII.

When you use **Serial.println()**, you send information formatted for the serial monitor. When you use **Serial.write()**, like in the application you are running now, you're sending raw information. Programs like Processing can understand these raw bytes.

Close the serial monitor. Run the Processing sketch by pressing the arrow button in the Processing IDE. Look at the Processing output window. You should see a list similar to the figure below.

The image shows a screenshot of the Processing IDE window titled "sketch\_150720a | Processing 2.2.1". The window has a menu bar with "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for running, stopping, and other actions. The main area shows a Java sketch with the following code:

```
logo = loadImage("https://www.arduino.cc/en/pub/skins/arduinoWide/size(logo.width, logo.height);

println("Ports série disponibles :\n");
printArray(Serial.list());

monPort = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
  if (monPort.available() > 0) {
    bgcolor = monPort.read();
    println(bgcolor);
  }
}
```

The output window at the bottom shows the following text:

```
Error opening serial port /dev/ttyS0: Incorrect serial port

Ports série disponibles :

[0] "/dev/ttyS0"
[1] "/dev/ttyS1"
[2] "/dev/ttyS2"
[3] "/dev/ttyS3"
[4] "/dev/ttyS4"
[5] "/dev/ttyS5"
[6] "/dev/ttyS6"
[7] "/dev/ttyS7"
[8] "/dev/ttyS8"

16
```

This is a list of all the serial ports on your computer. If you're using OSX, look for a string that says something like `"/dev/tty usbmodem411"`, it will most likely be the first element in the list. On Linux, it may appear as `"/dev/ttyUSB0"`, or similar. For Windows, it will appear as a COM port, the same one you would use when programming the board. The number in front of it is the **Serial.list[]** array index. Change the number in your Processing sketch to match the correct port on your computer.

Restart the Processing sketch. When the program starts running, turn the potentiometer attached to the Arduino. You should see the color behind the Arduino logo change as you turn the potentiometer. You should also see values printing out in the Processing window. Those numbers correspond to the raw bytes you are sending from the Arduino.



Once you have twisted and turned to your heart's desire, try replacing the pot with an analog sensor. Find something you find interesting to control the color. What does the interaction feel like? It's probably different than using a mouse or a keyboard, does it feel natural to you?



When using serial communication, only one application can talk to the Arduino at a time. So if you're running a Processing sketch that is connected to your Arduino, you won't be able to upload a new Arduino sketch or use the serial monitor until you've closed the active application.



With Processing and other programming environments, you can control media on your computer in some remarkable and novel ways. If you're excited about the possibilities of controlling content on your computer, take some time to experiment with Processing. There are several serial communication examples in both the Processing and Arduino IDEs that will help you explore further.

*Serial communication enables the Arduino to talk with programs on a computer. Processing is an open source programming environment that the Arduino IDE is based upon. It's possible to control a Processing sketch with the Arduino via serial communication.*