

Circuit Building and Programming with Arduino

Nick Marchuk
3/23/2018

A digital version of this document can be found at hades.mech.northwestern.edu

Tutorial Parts List:

[Storage box](#)

[Breadboard](#)

[6 colors of 24 gauge solid core wire](#)

[Wire strippers](#)

[Arduino Metro Mini](#) with pins already soldered on

[Micro USB cable](#)

([USB C to USB2 adapter](#))

[WS2812B breakout board](#)

[10k potentiometer](#)

[330Ω](#), [10kΩ](#) resistors

[Red](#), [green](#) LEDs

[Push button](#)

[Micro RC Servo](#) with header pins stuck in the plug

Software for the Arduino Integrated Design Environment (IDE):

Visit <https://www.arduino.cc/en/Main/Software> and download the installer for Arduino 1.8.5. Install it in the default location.

USB Driver to talk to Arduino Metro Mini:

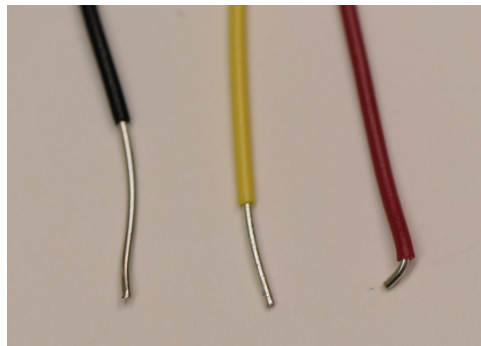
Visit <https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers> and select "Download VCP" for your operating system.

- On Windows, install it in the default location and reboot your computer.
- On Mac, run SiLabsUSBDriverDisk.dmg. Check your version of OSX (select the apple icon in the top left of your screen, go to About This Mac). If your version is 10.11 or higher, run Silicon Labs VCP Driver.pkg file. If your version 10.10 or lower, open the Legacy MacVCP Driver folder and run the Silicon Labs VCP Driver.pkg file. Install in the default location (you will probably need to enter your password at some point) and reboot your computer.

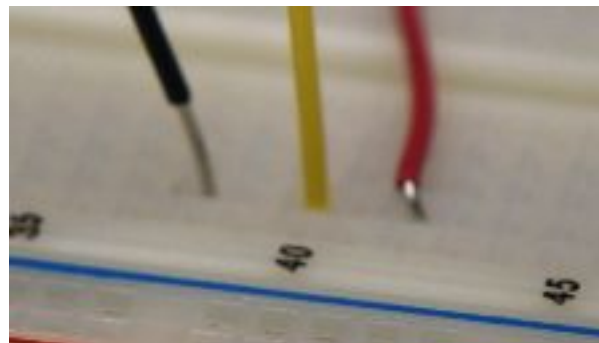
Electronics: Cut and strip a wire



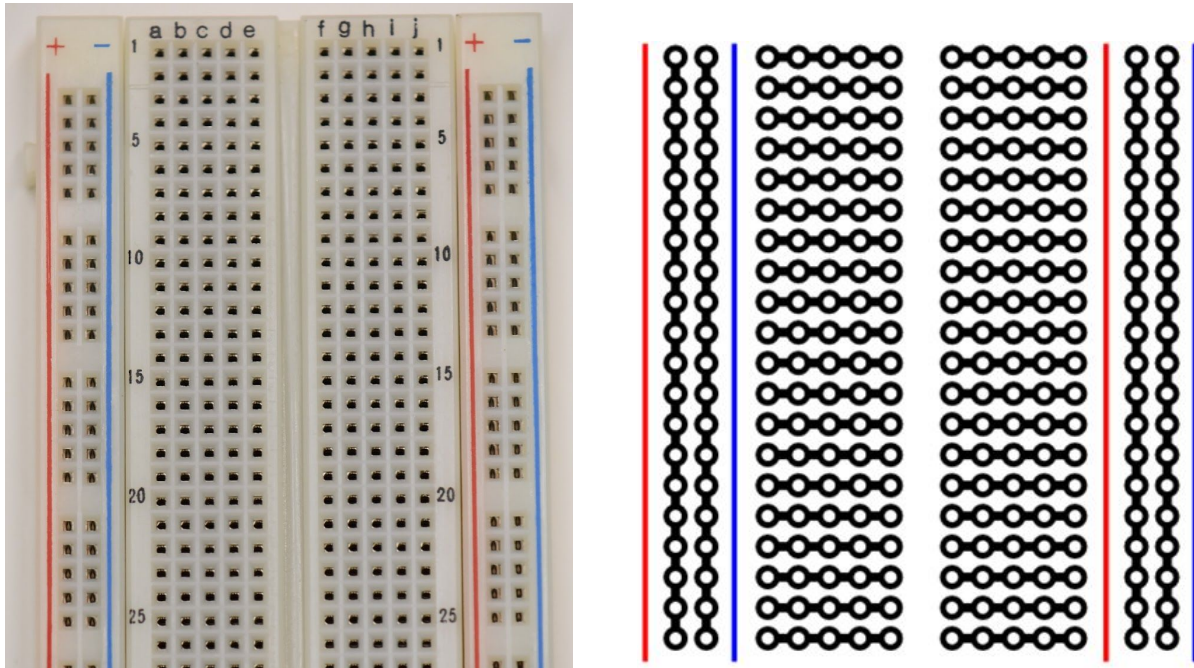
Practice using your wire strippers to cut a few 1" long wires. Remove about $\frac{1}{4}$ " of insulation from both ends of the wire, using the 26 AWG stripper hole. $\frac{1}{4}$ " of exposed wire is about the right length to be plugged into the breadboard without leaving any exposed wire to accidentally touch other exposed wires (the breadboard is about $\frac{1}{4}$ " deep).



Too much wire exposed (left), too little (right), goldilocks (center)

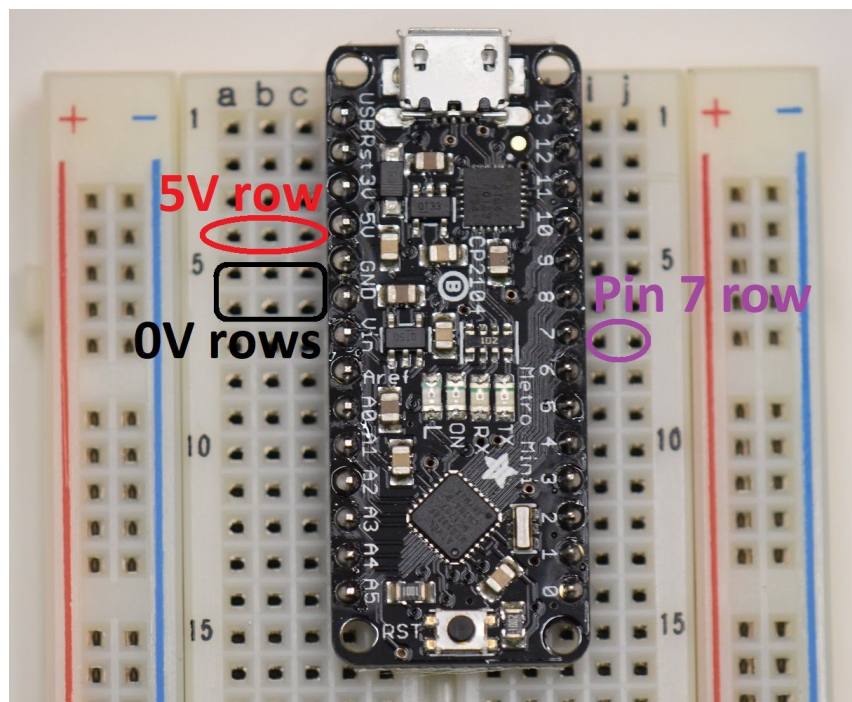


Electronics: How a breadboard works



A breadboard is a collection of conducting clips that make it easy to connect wires to components like resistors, LEDs, and Arduino pins. Along the outer edges of the breadboard are long columns of clips called rails. In the middle section of the breadboard are rows of 5 connected holes. The left and right side of the center channel are not connected.

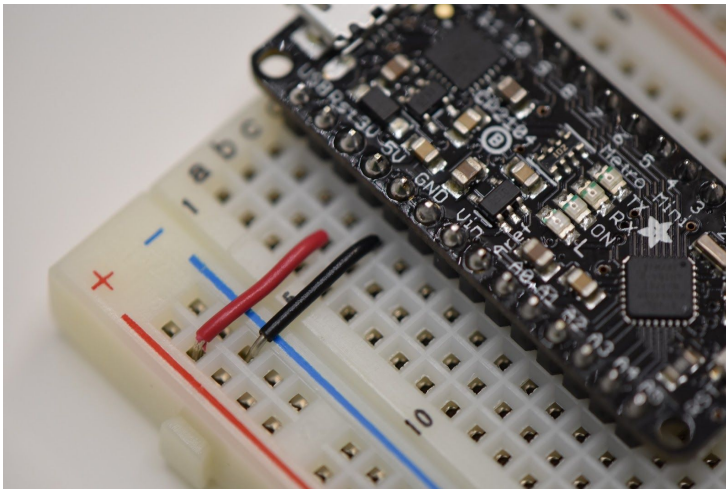
When a component like the Arduino Metro Mini is plugged into the breadboard, the rows become connected to the Arduino pins and give several access points to each pin.



Circuits usually need many access points to power and ground, so wire is used to connect the rails to the power rows to gain more power and ground holes. Make some wires to connect the left + rail to 5V and the left - rail to GND (0V). The color convention is to use **red wire** for 5V and **black wire** for Ground, so that you can quickly look at a circuit and debug it later.

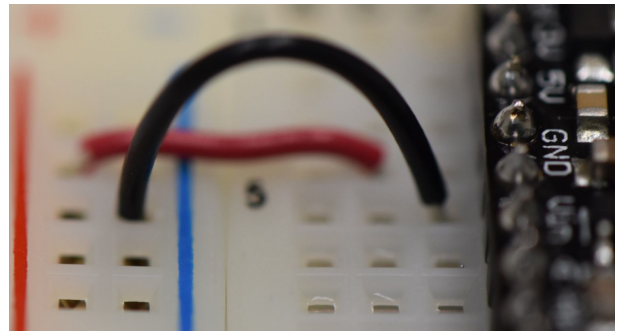
Note that the Arduino is supplied with 5V from the computer using the USB cable, and now the breadboard has access to the computer's 5V line. The computer would be very unhappy if you shorted it's 5V (no damage, but the USB port would be disabled and only reenabled by a reboot). Applying a voltage to the 5V connection, say from a 6V pack of AA batteries, would damage the computer's motherboard, so extra care should always be taken when using power from both the computer and an external source.

Bonus points for wires that are flush to the board.



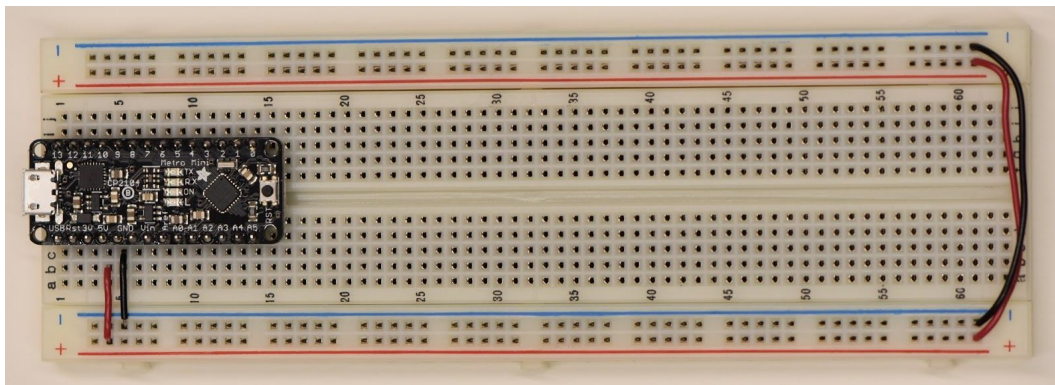
Flush

vs

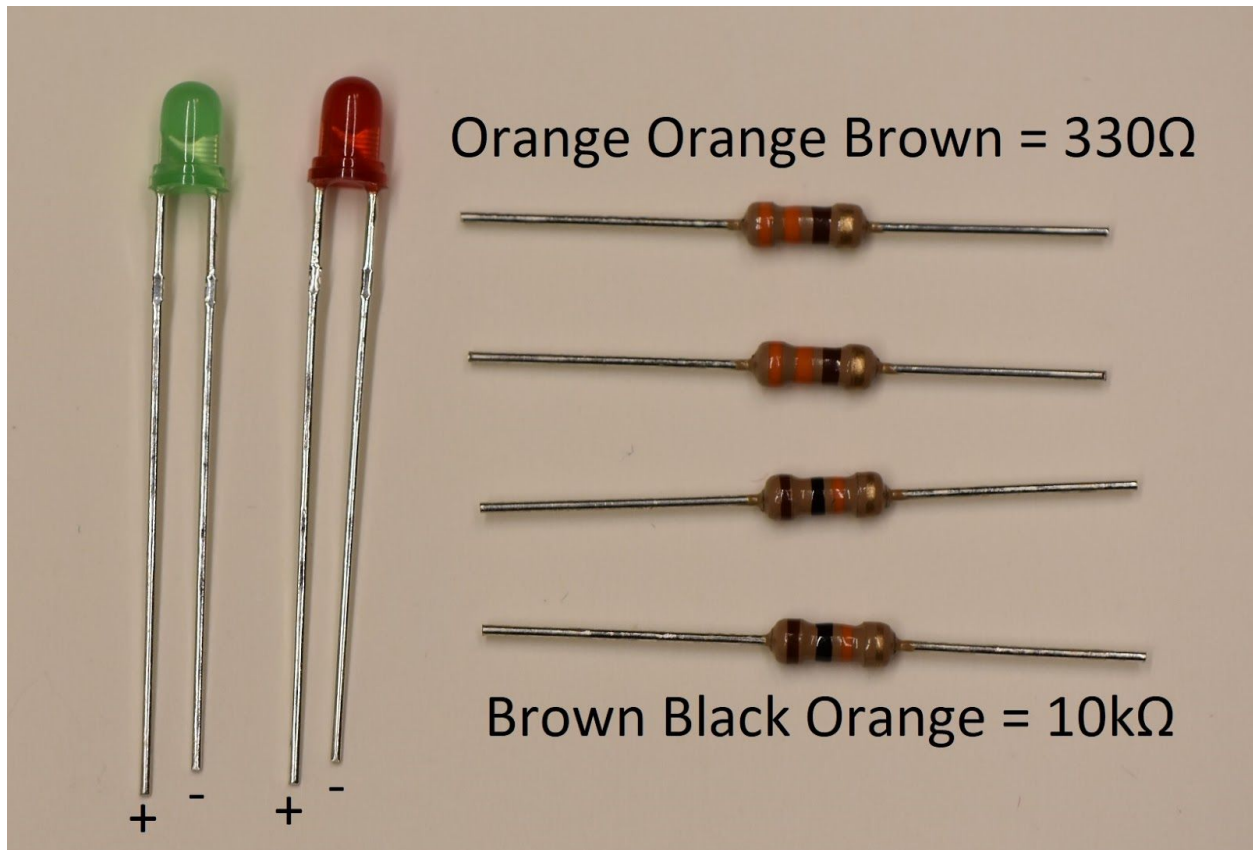


Loopy

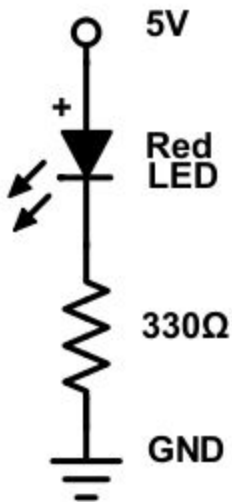
Extra bonus points for connecting the rails on the left to the rails on the right.



Electronics: Power an LED



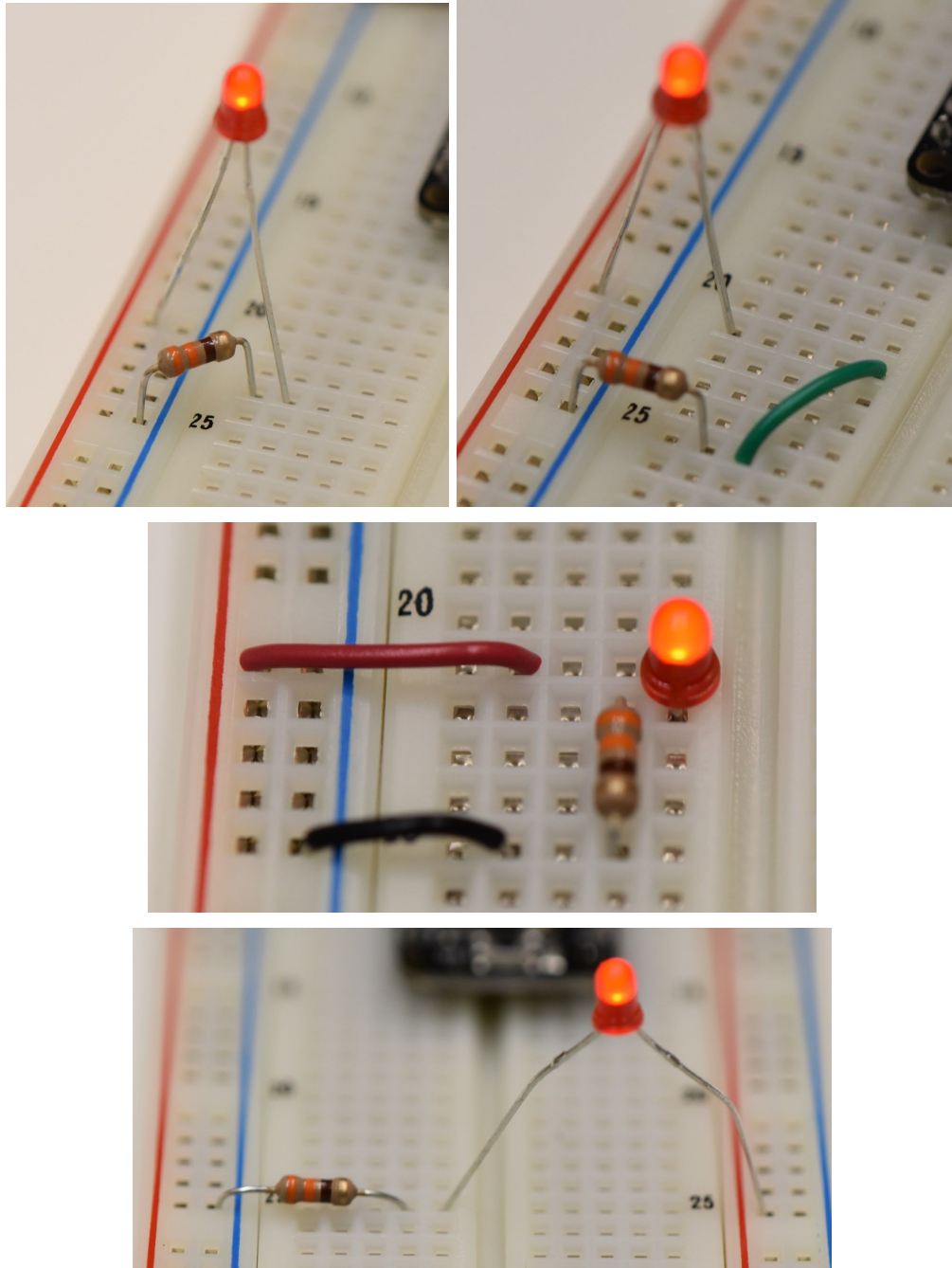
An LED will light when current flows through it, from the positive leg (longer wire) to the negative leg (shorter wire). A resistor in series with the LED is necessary to reduce the current through the LED and prevent it from overheating.



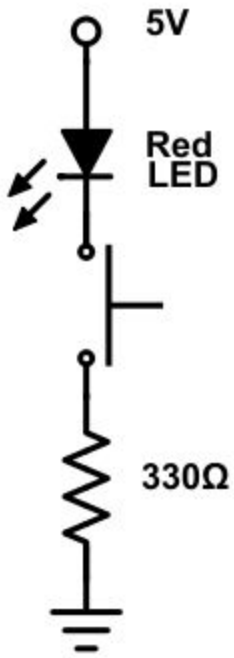
It is important to draw a circuit schematic before building the circuit. The schematic is the the reference for what you were supposed to build, and your guide when debugging.

Build this circuit somewhere on your breadboard. What happens when you replace the 330Ω resistor with the $10k\Omega$?

The schematic does not tell you how to make the connections on the breadboard; there are infinite ways to do that. Use whatever construction you are comfortable with! You can also trim the legs of the LED and resistor to make them less likely to fall out, but try to leave the positive leg of the LED a little longer so you remember which way is positive.



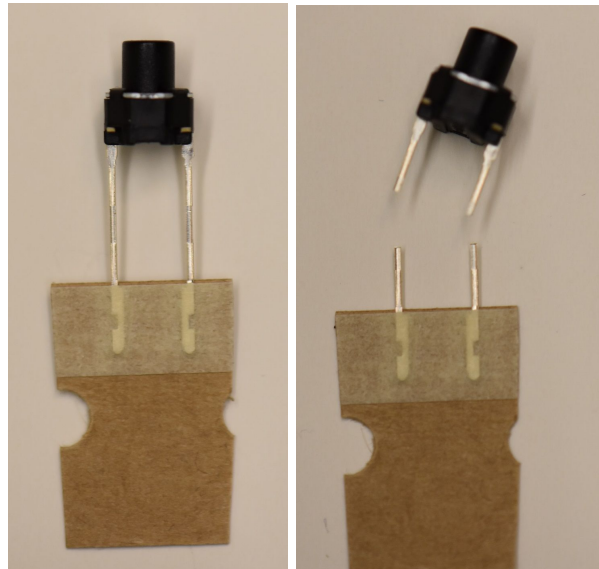
Electronics: Turn on an LED with a button



Edit your circuit so that the LED is on only when a push button is pushed.

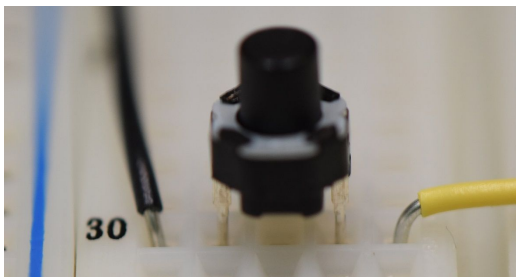
Do the order of the components matter?

How could you redesign the circuit so that the LED is on only when the button is not pushed?



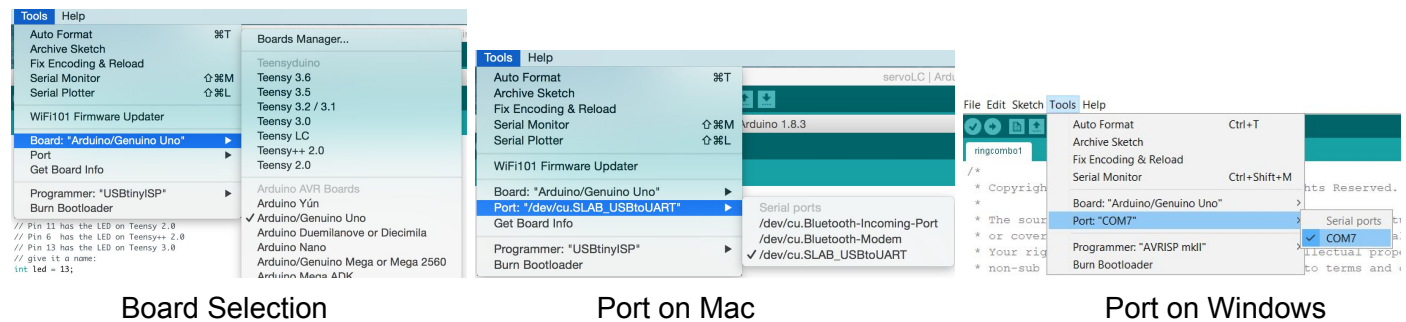
Note that the pushbutton has funny feet that do not fit in the breadboard. Cut them off!

Why does the following circuit not function?



Arduino programming: Blink an LED

Open the Arduino IDE software. Go to Tools→Board, and select Arduino/Genuino Uno. With the USB cable plugged in, go to Tools→Port, and note the available port names. Unplug the USB cable, and go back to Tools→Port. The port name that disappeared was your Metro Mini. Plug the USB cable back in, and select that port name.



Arduino programming uses the C/C++ language, with a few adaptations. When the Arduino board is powered, the function `setup()` is called. When `setup()` completes, `loop()` is called. When `loop()` ends, `loop()` is called again, until the board is

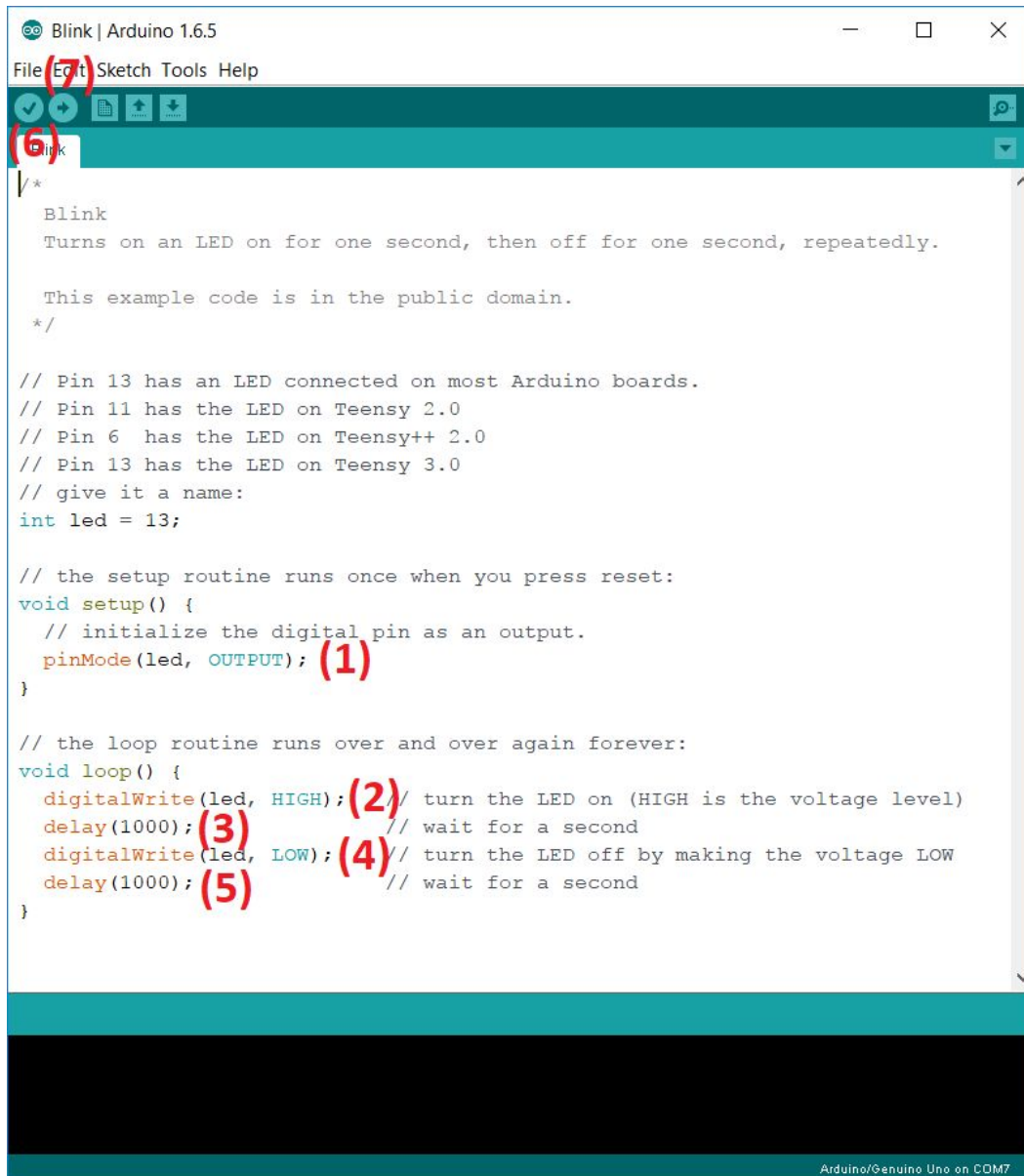
- unpowered,
- the reset button is pushed,
- or the board is reprogrammed from the IDE.

The program memory is nonvolatile, so it is not lost when power is removed, but the variables are volatile and are cleared.

Global variables, generally frowned upon in C programming, are common on microcontrollers like Arduino, and are usually defined above `setup()`. Line comments start with `//` and block comments appear between `/*` and `*/`.

```
sketch_mar22a §  
/*  
 * block comment  
*/  
  
// global variables up here  
  
void setup() {  
  // put your setup code here, to run once:  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
}  
  
|
```


The Arduino IDE contains many examples. Go to File→Examples→0.1Basics→Blink.



```
Arduino IDE: Blink | Arduino 1.6.5
File Edit Sketch Tools Help
Blink
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 *
 * This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// Pin 11 has the LED on Teensy 2.0
// Pin 6 has the LED on Teensy++ 2.0
// Pin 13 has the LED on Teensy 3.0
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT); (1)
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); (2) // turn the LED on (HIGH is the voltage level)
  delay(1000); (3) // wait for a second
  digitalWrite(led, LOW); (4) // turn the LED off by making the voltage LOW
  delay(1000); (5) // wait for a second
}

Arduino/Genuino Uno on COM7
```

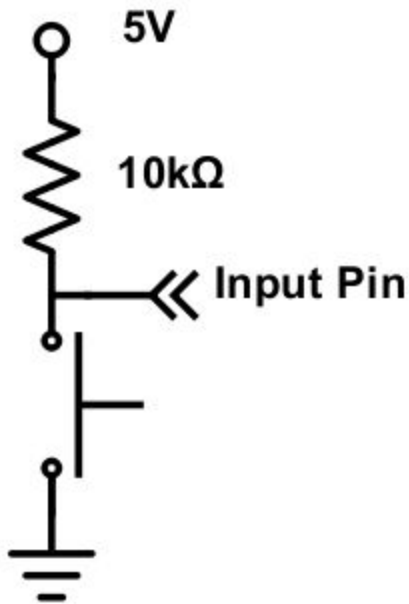
The Metro Mini board has an LED soldered to pin 13. The blink code (1) sets pin 13 as an output pin, (2) turns the pin on, (3) waits 1000 ms, (4) turns the pin off, (5) waits 1000 ms, and continues forever. Change the `delay()` calls to 100 ms, and (6) compile the code (check mark button). (7) Upload (rightwards arrow button) the code to the Metro Mini, and observe the new blinking rate. In the future, it is not necessary to compile the code before uploading, the upload button will do both.

Copy the code, make a new file, and paste the code in. Save it (the default location is /Documents/Arduino, leave it there). Choose a new pin (from 2-12), and edit the code to use the new pin. Build a circuit with an LED on that pin, and blink it at 4 Hz.

Draw your circuit diagram:

Arduino programming: Read a button and turn on an LED

Build the following push button circuit.



What voltage does the circuit output to the input pin when the button is not pressed? When the button is pressed?

Choose a pin to read the button voltage. In `setup()`, add a line that will initialize the pin as `INPUT`.

In `loop()`, declare a variable that will store the value of the button, like `int b;` and read the state of the pin into `b`:
`b = digitalRead(pin_number);` where `pin_number` is the number of the pin you plugged into.

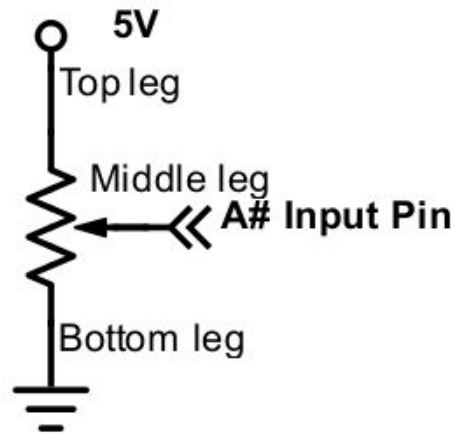
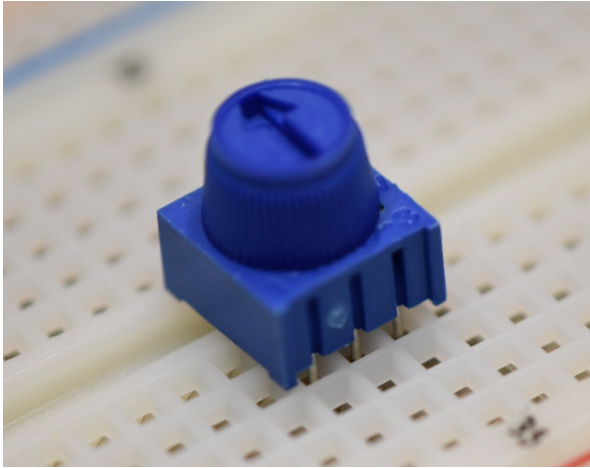
`digitalRead()` will return a 0 when the voltage is 0V and a 1 when the voltage is 5V.

After reading the button voltage into `b`, use an `if()` statement to determine if the button is pressed, and if it is, turn on the LED, and if it is not, turn off the LED.

Do you know the syntax for an `if()` statement in C? Check the Arduino reference guide at <https://www.arduino.cc/reference/en/>.

Arduino programming: Read a potentiometer and print the voltage to the computer

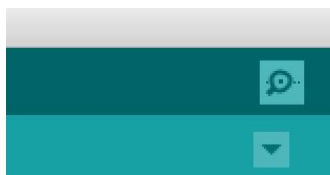
A potentiometer is a variable resistor knob with three legs. When the bottom and top legs are attached to Ground and 5V, the middle leg will output a voltage from 0V to 5V that is proportional to the angle of the knob. Potentiometers are usually limited to 180 degrees of rotation.



The Metro Mini can read voltages from 0V to 5V on pins that begin with A (A0 to A5). Build the potentiometer circuit and connect the output voltage to an A pin.

The function `analogInput(pin_name)` will return an integer from 0 to 1023 (0 for 0V, 1023 for 5V). The `pin_name` is the name of the pin you plugged into (example A0). The A pins do not need to be initialized with `pinMode()`. In `loop()`, declare an `int` variable, and set its value to the analog value of the potentiometer voltage. Change your `if()` statement so that if the potentiometer voltage is above 2.5V the LED is on, otherwise it is off. This is an example of 1 LED debugging!

The Metro Mini has the ability to turn the numeric value of a variable into text characters and print them to the computer. This occurs over USB, but is called Serial communication after the type of legacy functions the computer uses. To enable Serial communication, enable the Serial module in `setup()` by calling the function `Serial.begin(9600)`; In `loop()`, use the function `Serial.println(variable)`; to turn the value of `variable` into text and send the text characters to the computer. Upload the code, and open the Serial Monitor to see the data stream.



The Serial Monitor button in the top right of the IDE, looks like a magnifying glass

Turn the potentiometer to see the value change.

The Metro Mini is sending the value as fast as possibly can, which may upset your computer. Add a `delay()` at the end of `loop()` to slow the communication down to 10 Hz.

Also try the Serial Plotter in Tools→Serial Plotter.

Arduino programming: Read a potentiometer and set the brightness of an LED

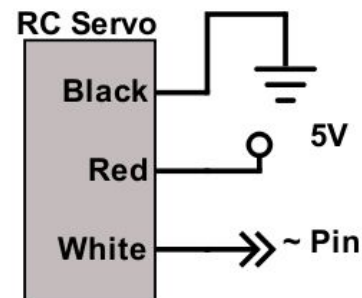
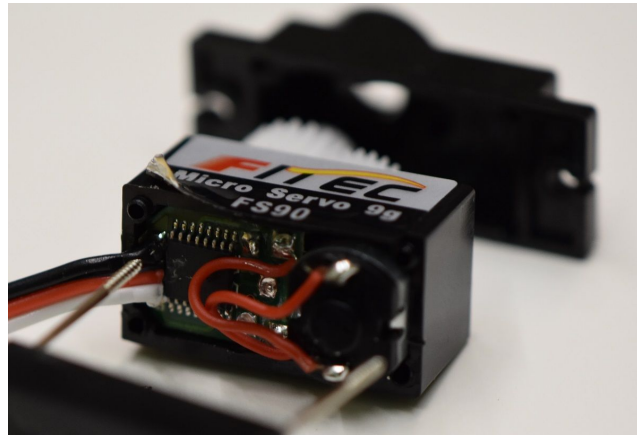
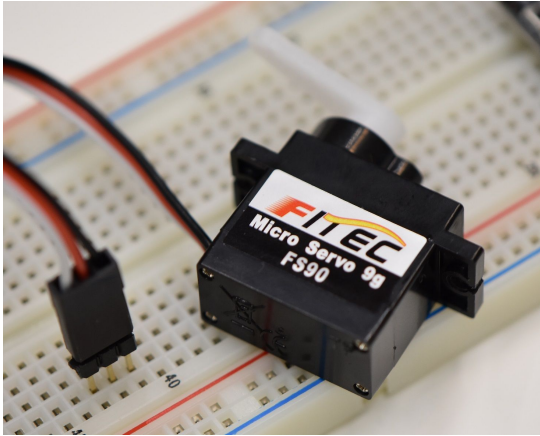
Most Arduino boards lack the ability to output analog voltages (voltages that are between 0V and 5V). There is a function called `analogWrite(pin, value)`, but it actually outputs a square wave at a high frequency, blinking the pin so fast that it appears to be analog (this is typically called pulse width modulation, or PWM). `value` can take integer values between 0 and 255, and only pins 3, 5, 6, 9, 10, and 11 can be called with `analogWrite()`.

Connect your LED circuit to a PWM capable pin and use `analogWrite()` to set the brightness of the LED proportional to the angle of the potentiometer.

- Did you draw a circuit diagram below?
- Did you remember `pinMode()` for your PWM pin?
- What is weird about the value you get from `analogRead()` and the value you use in `analogWrite()`?

Arduino programming: Using an internal library to control the position of an RC servo

A Remote Control style Servomotor is an inexpensive way to get position control of a motor. The RC servo has a built in controller, brushed DC motor, gears, and potentiometer. It reads its own position and forces it to match the commanded position told to it by the Arduino.



Connect the black wire of the servo to Ground and the red wire to 5V. The white wire goes to a PWM capable pin on the Arduino. Larger RC servos can draw more current than USB can supply and need an external power supply, usually 4.8-6V.

The voltage signal that commands a desired angle to the servo uses a library, like the way Serial communication works. The servo library is not included in the code by default, but it is preinstalled with the IDE, so you can use it by adding `#include <Servo.h>` at the top of your code. Declare a global variable to represent the servo above `setup()`, like `Servo myServo;` In `setup()`, assign the servo variable the PWM pin you connect the servo to, like `myServo.attach(pin_number);`

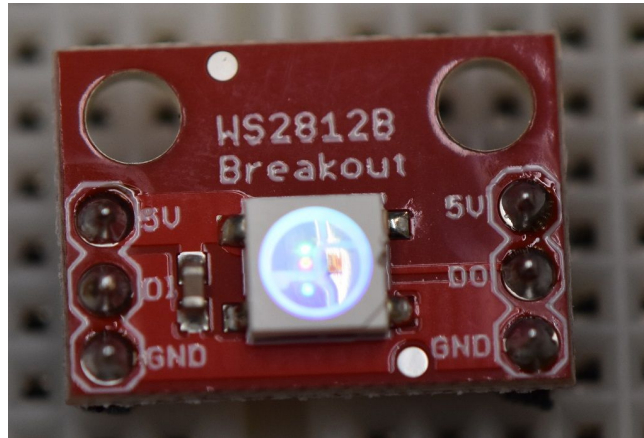
You can set the position of the servo by using `myServo.write(angle)`, where `angle` is an integer from 0 to 180, representing 0 degrees to 180 degrees.

Edit your code so that when the button is pushed, set the angle of the servo proportionally to the angle of the potentiometer. Having trouble? Print the value of the angle to the Serial Monitor to check your math.

Try holding down the button and turning the potentiometer, and try turning the potentiometer and tapping the button. How fast can the motor move?

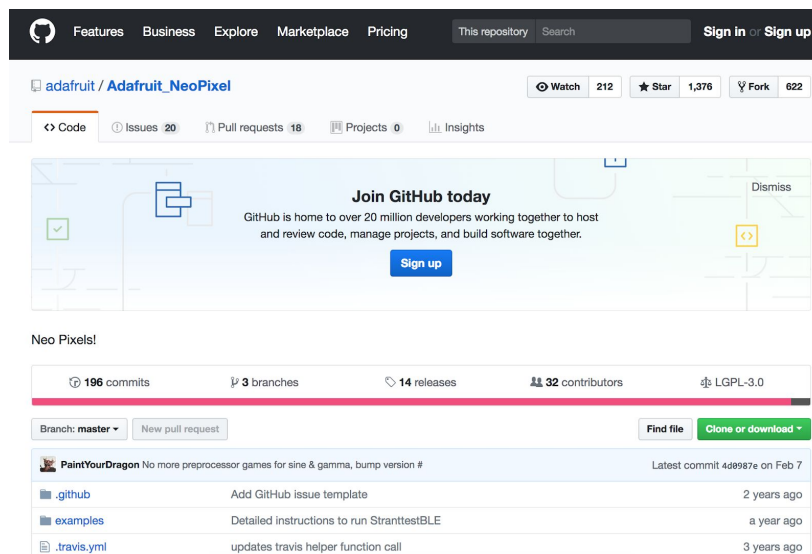
Arduino programming: Using an external library to control the color of an RGB LED

The WS2812B is an addressable RGB LED that is commonly known as a NeoPixel. Using one pin from the Arduino, you can control the brightness and color of hundreds of NeoPixels. We only have one, but you can purchase NeoPixels on flexible tape, stiff lines, arcs, or panels, or a variety of different configurations, and make cool displays and lighting effects.

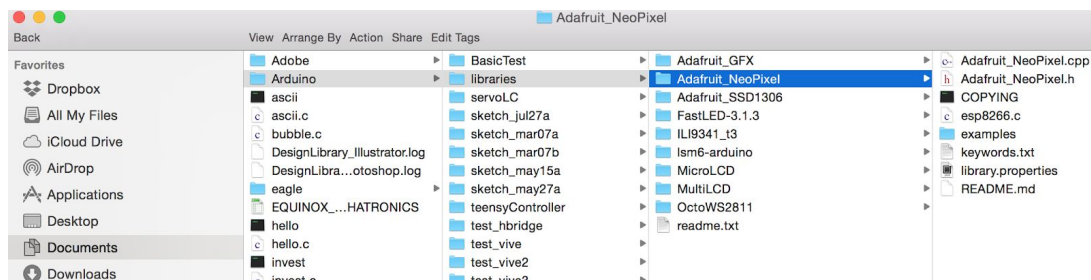


The NeoPixel requires a connection to 5V, Ground, and an Arduino pin to the WS2812B DI pin. If you had more NeoPixels, DO of the first NeoPixel would be connected to DI of the second, and so on.

The NeoPixel library does not come with the Arduino IDE, so go to https://github.com/adafruit/Adafruit_NeoPixel and download the library (from the green “Clone or download” button).



Unzip the folder, rename it to Adafruit_NeoPixel, and move it to /Documents/Arduino/libraries (you may have to make a folder called libraries in the Arduino folder).



Close and reopen the Arduino IDE so that it updates the available libraries. You can go to File→Examples→Adafruit_NeoPixel to see the example code that comes with the library. Sometimes external libraries downloaded from the internet are well documented and come with several different examples, but sometimes the documentation is sparse.

Rather than use an example, get control over the NeoPixel in your code by doing the following:

- Include the library at the top: `#include <Adafruit_NeoPixel.h>`
- Create a global variable to represent the NeoPixel: `Adafruit_NeoPixel pixel = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);` where `NUMPIXELS` is 1 and `PIN` is the pin you connected to DO.
- Initialize the variable in `setup()` using `pixel.begin();`
- Set the color in `loop()` using `pixel.setPixelColor(0, pixel.Color(RED, GREEN, BLUE));` where `RED`, `GREEN`, and `BLUE` are integers from 0 to 255 (off to bright) and then calling `pixel.show();`

In the same place where you set the position of the servo, set the brightness of the red portion of the LED proportional to the potentiometer angle.

Still have time left? Edit the code so that:

- The NeoPixel outputs a color proportional to the angle of the potentiometer, initially red
- Every time the button is pushed, toggle the color of the NeoPixel (red to green to blue to red, ...)
 - This is trickier than it sounds. You will need a variable to remember the current color of the NeoPixel, and an `if()` statement that can detect if the button was just pressed, as opposed to is currently pressed.