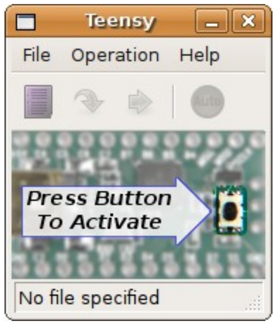
Tutorial 1: Software Setup

To use your Teensy or Teensy++ with the Arduino IDE, you will need to install a few programs.

Install Teensy Loader

The Teensy Loader application downloads programs to your Teensy board and lets you run them. When you use Arduino, this will be done automatically, but doing it manually is a simple way to check that your board works.

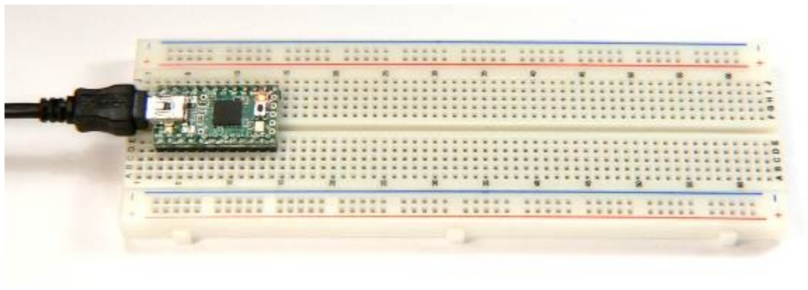
Just download the Teensy Loader and the LED Blink samples. No special installation is needed, just double click on the Teensy Loader to run it. You should see this window:



If you are using Linux, the 49-teensy.rules file must be copied to /etc/udev/rules.d/ to give non-root users permission to access the Teensy USB device.

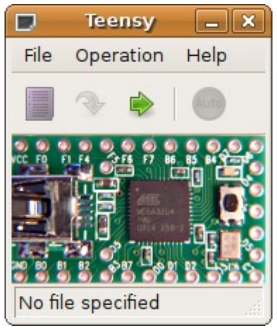
Download LED Blink

Now you are ready to connect your Teensy. Plugging the Teensy into your breadboard will protect its pins and give you a solid base that doesn't move easily when you touch the cable. Then just plug in the USB cable and the LED should blink slowly.



When the Teensy boots up, it always runs whatever program has been loaded. All new Teensy boards come with the slow LED blink pre-loaded.

When you plugged in the cable, nothing happened with the Teensy Loader. You need to press the pushbutton to put your Teensy into programming mode. When you press the button, the window Teensy Loader should show your board.



Now you can load your Teensy with a different program. Use the File->Open menu to open blink_fast.hex (available from the page where you downloaded the Teensy Loader). The bottom line will tell you which file is open and how much of the flash memory it uses.

Just click the "Program" button to write this new code to your Teensy, and then click the "Reboot" button to cause it to run. Your Teensy should be blinking rapidly. You have verified everything works and you're ready to use it with Arduino.

Install Arduino

The next step is to download and install the Arudino Software.

On Windows and Linux, the software is simply a large ZIP file. You will need to extract it.

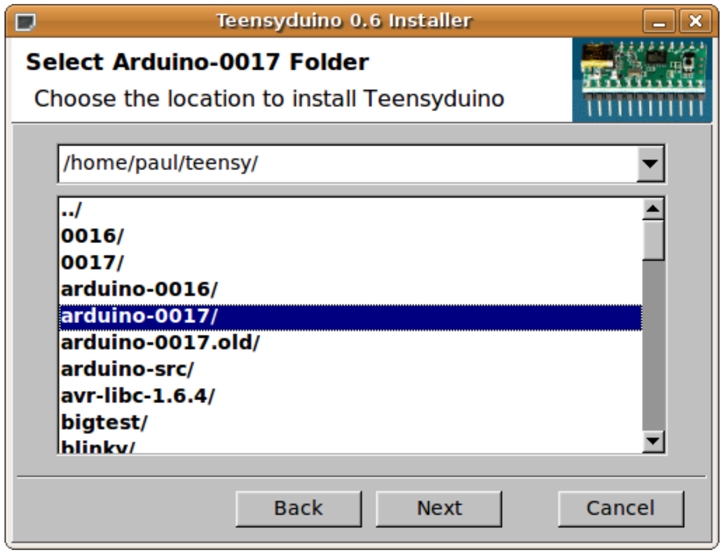
On Macintosh, starting with version 0017, the software is a disk image containing the program. You must copy it to /Applications or your home directory.

Remember the location where you extracted or copied the Arduino software, since you will need it shortly.

Install Teensyduino

The Arduino software does not come with support for the Teensy, so you must run the Teensyduino installer to add the Teensy files to your Arduino software.

The installer will only ask 1 question, the location of your Arduino software. The "Next" button will only activate until a directory containing the Arduino software is selected.

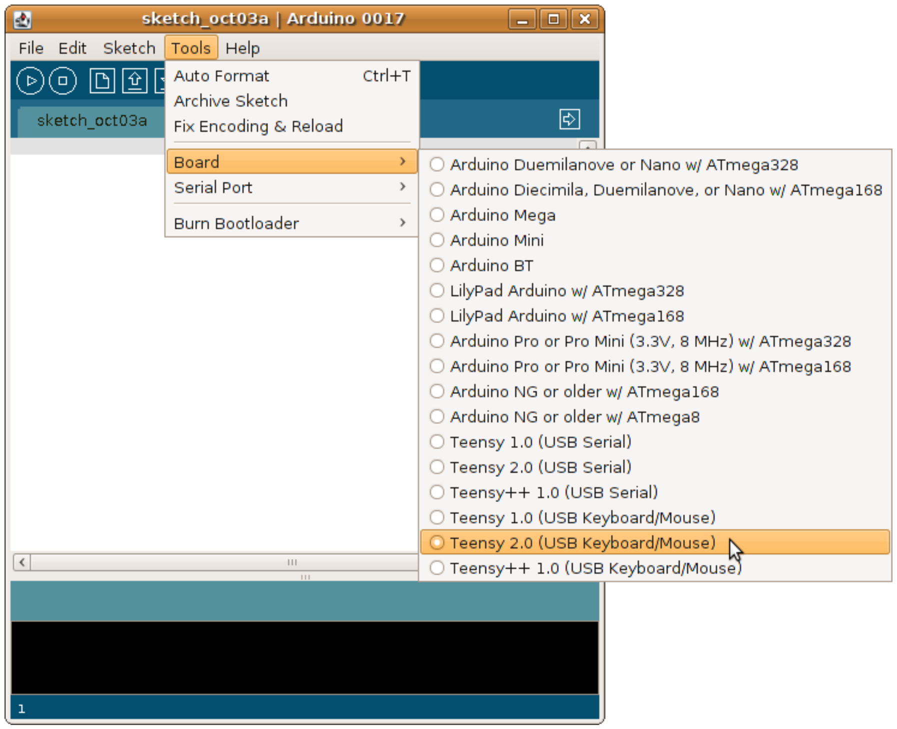


Just continue clicking "Next" until the installation is completed.

If you are using Windows, you should also run this Windows Serial Installer. If you choose the "USB Serial" option, this will allow the Windows Found New Hardware Wizard to properly find the driver.

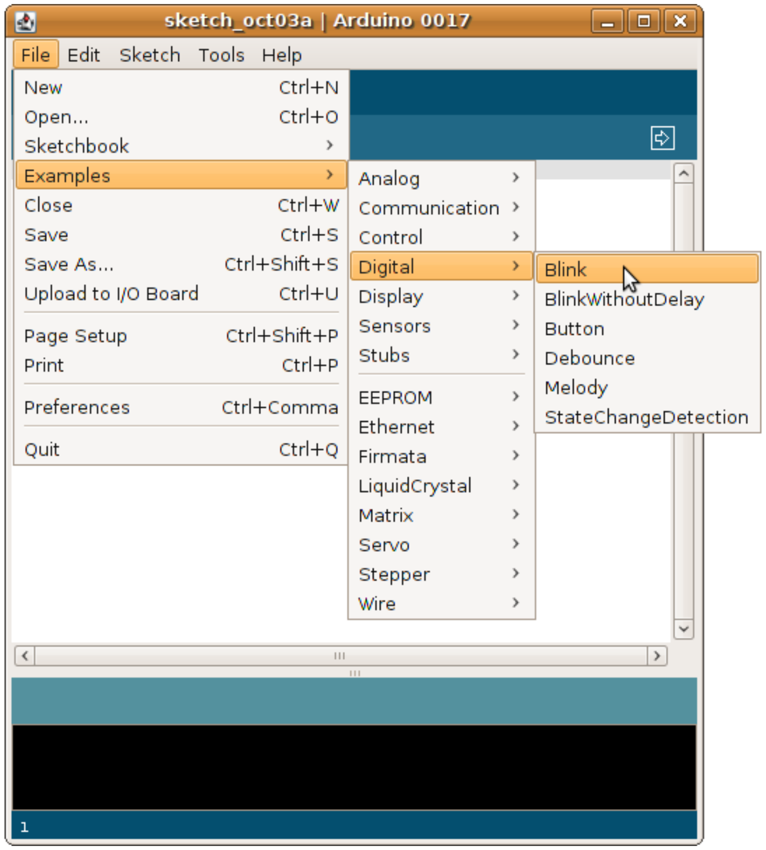
Start Arduino & Choose The Board

Now you are ready to run Arduino. The first step is to select the board you will be using, from the Tools->Boards menu. This menu will have entries for Teensy because you ran the Teensyduino installer.


sketch_oct03a | Arduino 0017
File
Edit
Sketch
Tools
Help
sketch_oct03a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Board
Serial Port
Burn Bootloader
Arduino Duemilanove or Nano w/ ATmega328
Arduino Diecimila, Duemilanove, or Nano w/ ATmega168
Arduino Mega
Arduino Mini
Arduino BT
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
Teensy 1.0 (USB Serial)
Teensy 2.0 (USB Serial)
Teensy++ 1.0 (USB Serial)
Teensy 1.0 (USB Keyboard/Mouse)
Teensy 2.0 (USB Keyboard/Mouse)
Teensy++ 1.0 (USB Keyboard/Mouse)
1

Open LED Blink Example

The Arduino Software comes with many examples. Use the File->Examples->Digital->Blink menu to open the LED blink example.

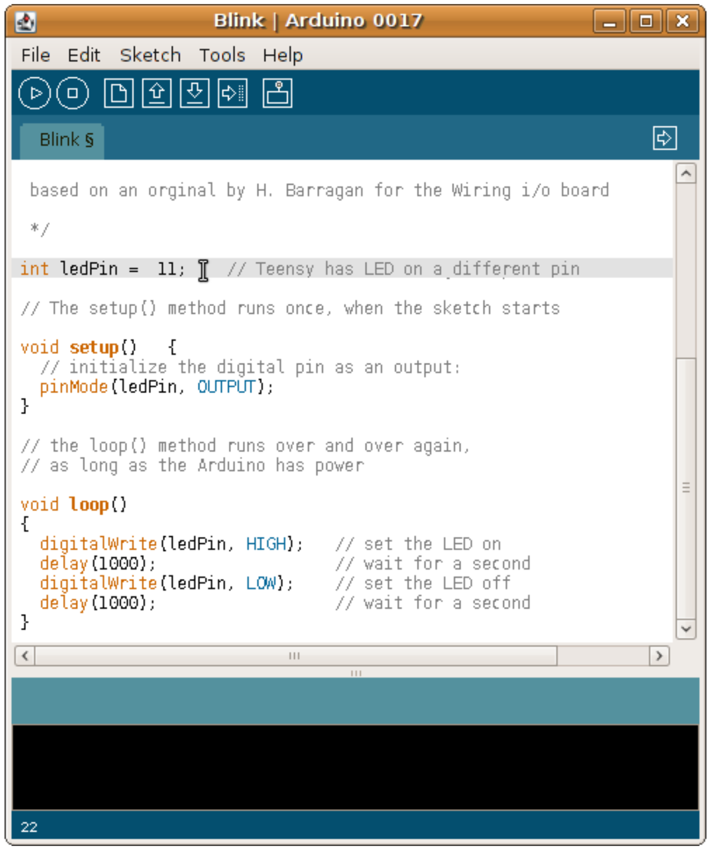


Edit pin number

The Teensy has its LED connected to a different pin, so you will need to change the pin number. Find the line "int ledPin = 13;" and change the number.

The Teensy 2.0 board (ATMEGA32U4 chip) has the LED on pin 11.

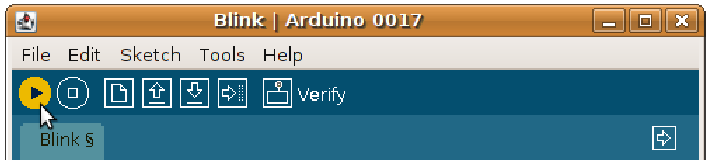
The Teensy++ 1.0 (AT90USB646 chip) and Teensy 1.0 board (AT90USB162 chip) have the LED on pin 6.



```
 based on an orginal by H. Barragan for the Wiring i/o board

 */

int ledPin =  11;  // Teensy has LED on a different pin

// The setup() method runs once, when the sketch starts

void setup()   {
  // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);
}

// the loop() method runs over and over again,
// as long as the Arduino has power

void loop()
{
  digitalWrite(ledPin, HIGH);   // set the LED on
  delay(1000);                  // wait for a second
  digitalWrite(ledPin, LOW);    // set the LED off
  delay(1000);                  // wait for a second
}
```

Compile and Download

To compile this code, click the "Verify" button.



A message will appear showing the compiled size. The Teensy Loader will automatically update with the new file.

Just press the button on the Teensy to program the code.

You can try changing the delay times. Just recompile and download again. When the Teensy is running a previously loaded program, you can also use the Upload button.

Now that you have the software set up, you're ready to begin attaching circuits to the Teensy and writing your own code to control them!

Tutorial 2 will show you how to connect and use a RGB LED.

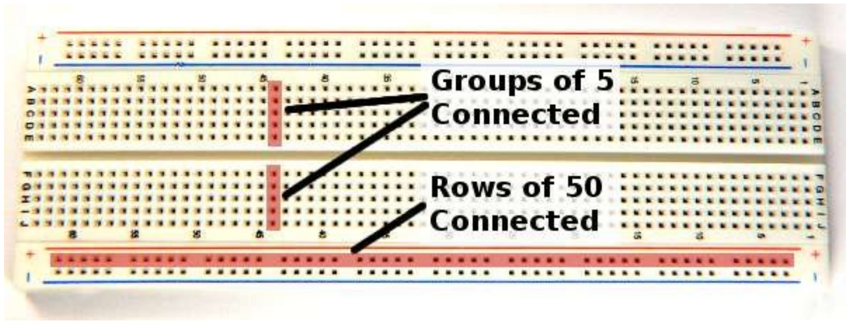
Tutorial 2: RGB LED

In this tutorial, you will connect a RGB (Red, Green, Blue) LED (Light Emitting Diode) to your Teensy. Because there are 3 different color LEDs inside, you can turn on different combinations to simulate many colors. By using more sophisticated programming, you can achieve very interesting color effects.

Breadboard Usage

A solderless breadboard gives you a very quick and easy way to build circuitry. If you have used a breadboard before or are familiar with how it works, you can skip this section.

Inside the breadboard, groups of holes are connected, so when you plug wires or components into holes from the same group, an electrical connection is made.



Most of the holes are in groups of 5. Usually you will connect components by plugging them into nearby groups of 5, and then add wires to connect those groups to others to complete your circuit.

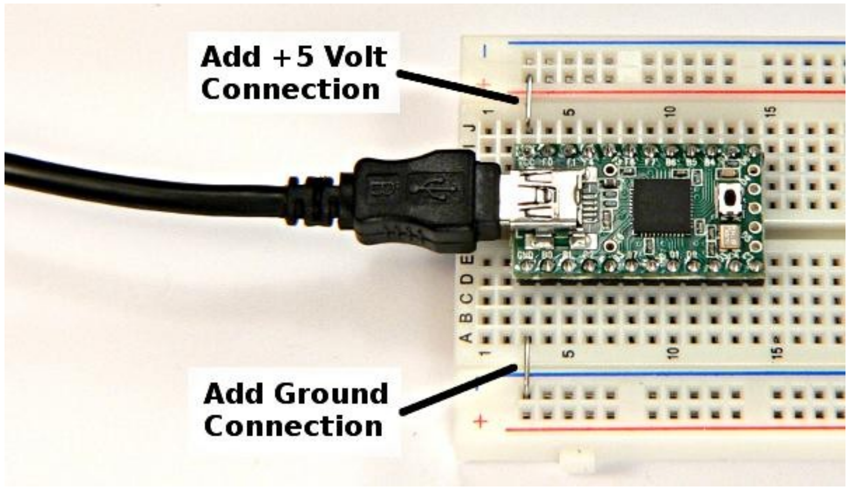
Along the top and bottom of your breadboard are long groups which are usually used for power. Because many connections are made to the 2 power lines, it's very convenient to have them run the entire length of the breadboard.

How you arrange the wires is largely a matter of personal style. In these photos, power connections and most components are mounted with legs bent at 90 degree angles and trimmed short. Usually you don't change these, so spending a few extra moments to mount these close keeps them out of your way. Often longer run connections are changed, so leaving extra length makes experimenting easier.

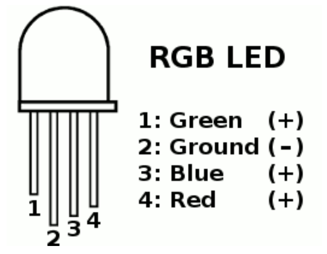
The electricity does not care if you carefully measured a short wire or used a (reasonably) long wire. Use whatever style you like!

LED Wiring & Testing

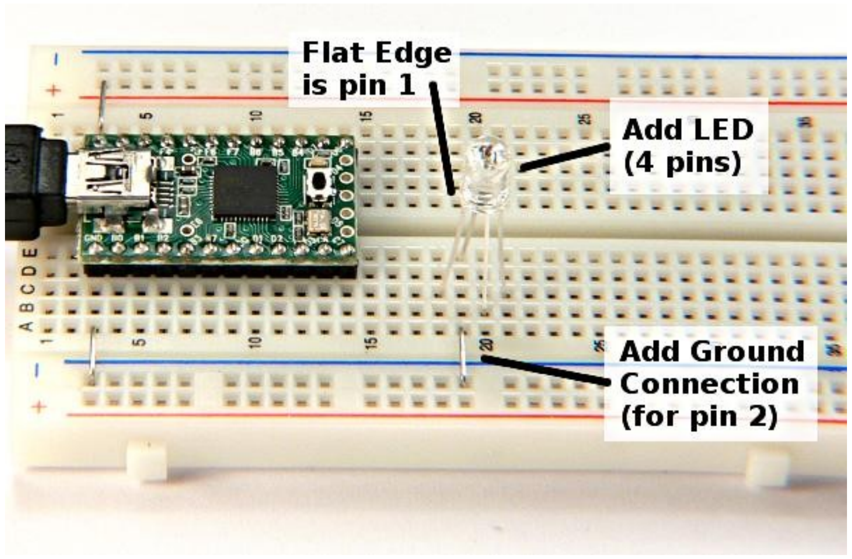
The first step is to add short wires to connect the long power rows to the power provided by the Teensy, which comes from the USB cable.



Next you will need to plug the RGB LED into your breadboard. There are 4 pins, where pin 1 is the shortest and located on the side with the flat edge.

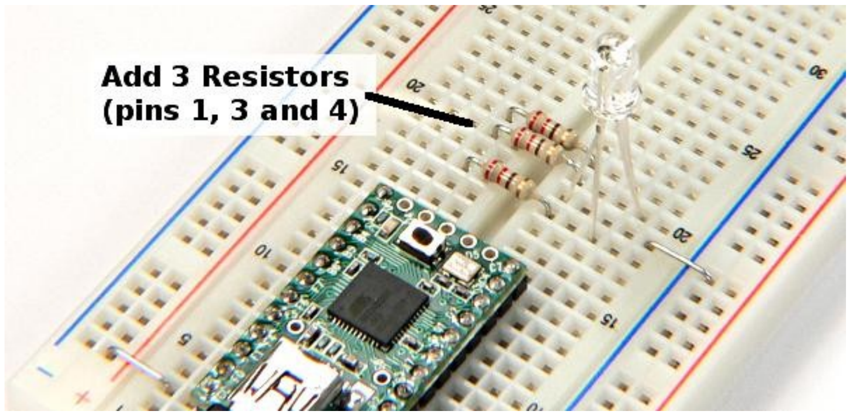


You will need to spread the pins slightly so they fit into 4 separate groups of holes. Pin 2 needs to be connected to ground, so place a short wire between that hole group and the ground row.



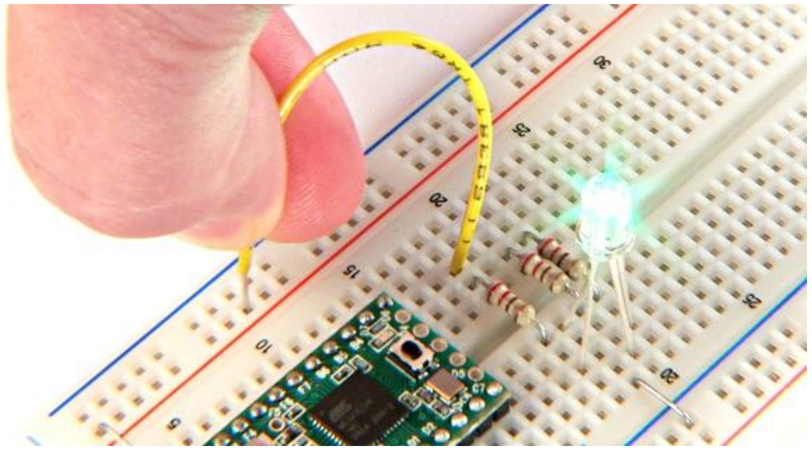
On each of the 3 positive pins, place a 220 ohm resistor (red, red, brown, gold).

Unlike a light bulb, the diode inside a LED will use as much power as it can. You must connect a resistor which will serve to limit the power. Never connect a LED directly to the power supply. Doing so would destroy the LED.



Now you are ready to test if the LED works. It is always good to test your connections (if possible) before you attempt to make them work from code in the Arduino software.

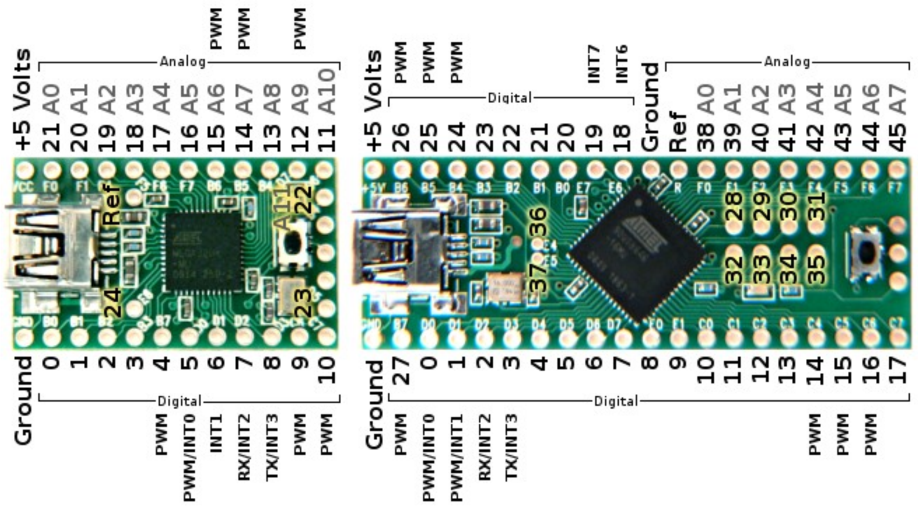
Simply touch a piece of wire between the resistor to the +5 volt power row. The LED should light!



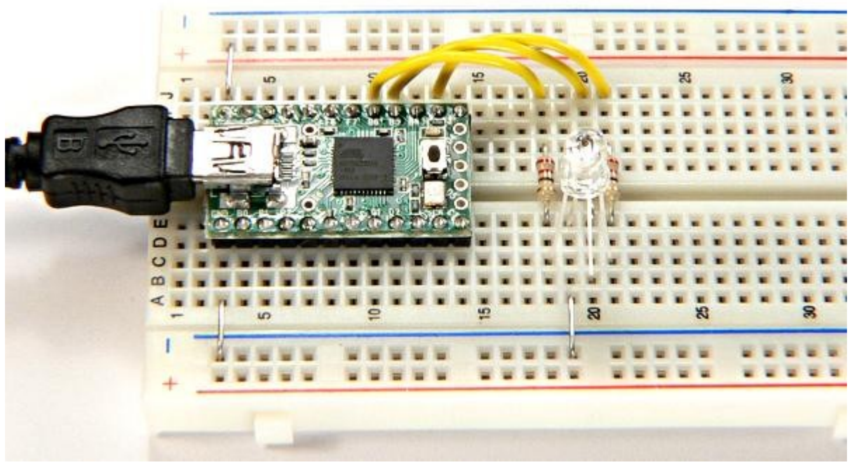
Repeat this test for the Blue and Red. Remember, do not touch the power directly to the LED. A resistor must always be connected between the power and LED.

Connect to Teensy Pins

With the LED ready, all that's left is connecting the 3 resistors to 3 pins on the Teensy. There are many pins to choose from! You should pick 3 pins with the **PWM** feature. Here are the Teensy pinout diagrams (and the old Teensy 1.0 diagram).

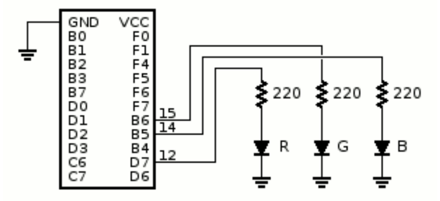


Using the Teensy 2.0 board, pins 12, 14 and 15 have PWM and are close. Just plug in 3 wires to connect from the resistors to the pins.



Schematic Diagrams

Electronic circuits are documented using schematic diagrams. Here is a simple schematic for the connections you just made.



From this schematic, it is easy to see which color is connected to which pin. Schematics document the connections, without details of how the actual connections are made.

When you find information about how to connect more types of circuit to your Teensy, usually they will be expressed in schematic diagrams.

Using Digital Output

To start using the RGB LED, open the blink example, from the File->Examples->Digital->Blink menu. Like in the first tutorial, change the pin number to 12 (or whatever pin has the red LED), compile and load it. You should see the LED blinking red.

Before continuing, you should quickly repeat for the other 2 pins. If any wire is not connected, you can much more easily diagnose and fix the problem by testing each color individually.

The blink example contains 2 sections, a "setup" function that configures the pin using "pinMode", and a "loop" function which turns the pin HIGH and LOW. You can easily expand this program by copying these things 3 times, and change "ledPin" to a name for each separate pin.

You can open these examples and copy-and-paste from your browser into Arduino

```
int redPin =  12;
int greenPin =  15;
int bluePin =  14;

// The setup() method runs once, when the sketch starts

void setup()   {
  // initialize the digitals pin as an outputs
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

// the loop() method runs over and over again,

void loop()
{
  digitalWrite(redPin, HIGH);
  delay(500);
  digitalWrite(greenPin, HIGH);
  delay(500);
  digitalWrite(bluePin, HIGH);
  delay(500);
  digitalWrite(redPin, LOW);
  delay(500);
  digitalWrite(greenPin, LOW);
  delay(500);
  digitalWrite(bluePin, LOW);
  delay(500);
}
```

When you run this program, the LED will be red, then when the green LED turns on, you will see yellow, because red and green light combine to form yellow. When the blue pin is turned on, you should see white (perhaps with some color if the 3 LEDs and resistors are not perfectly matched). Then when the red turns off, you should see cyan, and when the green turns off, you should see blue. The pattern will keep repeating, because "loop" continues to run over and over.

You can experiment with turning on different combinations of colors and using different delay times. Some interesting patterns are possible, but to really achieve interesting colors and effects, you need more control than just turning each color completely on or completely off.

Using Analog (PWM) Output

The PWM pins have the ability to turn on at different intensity, from 0 (fully off) to 255 (fully on). PWM stands of Pulse Width Modulation, which means the pin is actually pulsing on and off very rapidly to make this happen, but the net effect is you can control the brightness of each color.

To control the intensity, just use "analogWrite" in place of "digitalWrite". The second parameter should be a number between 0 to 255, instead of only "LOW" or "HIGH". Here is one example:

```
int redPin =  12;
int greenPin =  15;
int bluePin =  14;

void setup()   {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop()
{
  analogWrite(redPin, 30);
  delay(500);
  analogWrite(greenPin, 200);
  delay(500);
  analogWrite(bluePin, 40);
  delay(500);
  analogWrite(redPin, 150);
  delay(500);
  analogWrite(greenPin, 0);
  delay(500);
  analogWrite(bluePin, 250);
  delay(500);
}
```

With analogWrite you can create almost any color by changing the numbers!

Using Variables

Your program already contains 3 variables, "redPin", "greenPin", and "bluePin". These are assigned a number and never changed, because it would be senseless to change the number when the wire remains physically connected to the same pin.

You can create more variables and change them as your program runs. Here is an example which uses a variable to fade the LED color slowly from green to red.

```
int redPin =  12;
int greenPin =  15;
int bluePin =  14;

void setup()   {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

int redIntensity = 0;

void loop()
{
  // set all 3 pins to the desired intensity
  analogWrite(redPin, redIntensity);
  analogWrite(greenPin, 255 - redIntensity);
  analogWrite(bluePin, 0);

  // remain at this color, but not for very long
  delay(10);

  // increase the red
  redIntensity = redIntensity + 1;

  // since 255 is the maximum, set it back to 0
  // when it increments beyond 255
  if (redIntensity > 255) {
    redIntensity = 0;
  }
}
```

In the previous examples, every time "loop" ran, it did the same thing. But because the action now depends on a variable, each time "loop" runs it will create a different color. The delay is reduced to a very short time, so the loop runs 100 times per second. However, the color change is very small each time, so the LED smoothly fades from green to red.

In the analogWrite to the green pin, the red intensity is subtracted from 255, so when red is off, green is on, and as red increases, green will decrease. Each time the loop runs, "redIntensity" increases.

Near the end of loop is an "if" condition. This causes the following code between the curly braces to only run if the condition is true. When "redIntensity" becomes too large, it is set back to 0. The next time the loop runs, the LED will instantly go back to green. When "redIntensity" is not greater than 255, the code is not run, so it is not set back to zero.

You can create as many variables as you like (within the memory capacity of the chip) and use them in almost any way to achieve different color effects.

Now that you can create output to the pins, you are ready for Tutorial 3 to receive input signals.

Tutorial 3: Serial Monitor & Input

In this section, you will use the serial monitor to see more information about what your program is doing. You will also use pins in input mode to receive signals, which allows your code to respond to real world events!

Serial Monitor & Serial.print

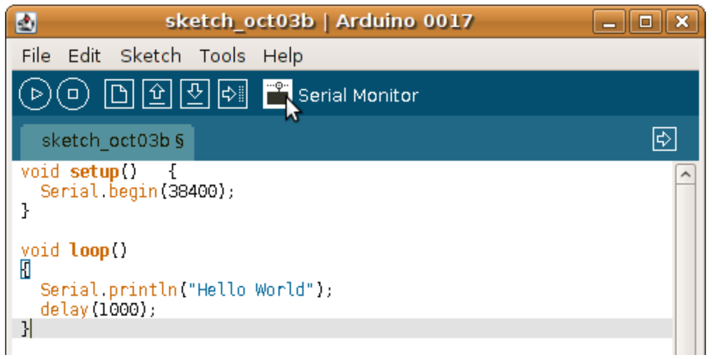
You can send messages to the Serial Monitor window on your PC or Mac, which is very helpful to see what your program is doing. Here is a simple example.

```
void setup()   {
  Serial.begin(38400);
}

void loop()
{
  Serial.println("Hello World");
  delay(1000);
}
```

Inside the setup function, "Serial.begin(38400)" initializes the communication. The number 38400 is the baud rate. The Teensy always uses USB speeds, so this number doesn't matter.

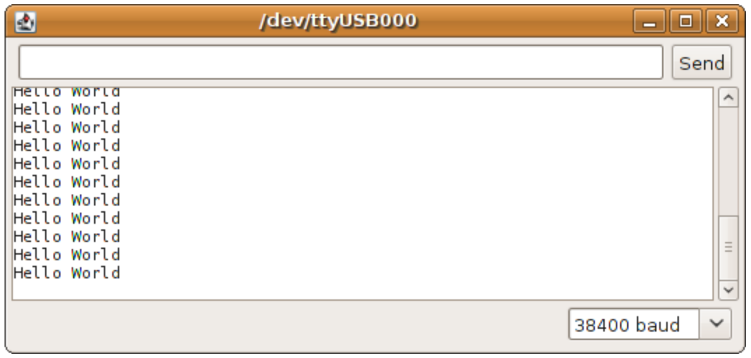
Then you can use Serial.print and Serial.println to send information. This example just sends "Hello World" every second.

To actually see the messages, click the Serial Monitor button.



```
void setup()   {
  Serial.begin(38400);
}

void loop()
{
  Serial.println("Hello World");
  delay(1000);
}
```

The Serial Monitor window should appear, with these messages printing as your program runs.

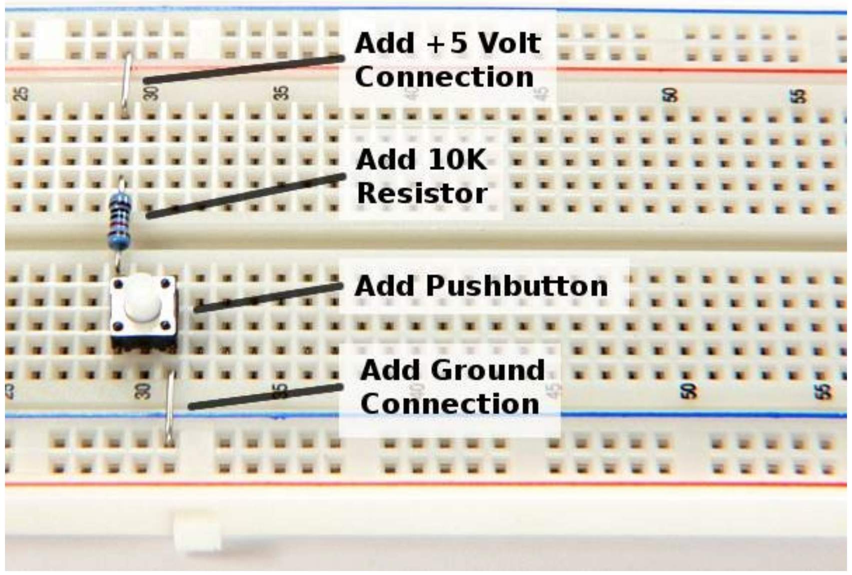


```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Simply printing "Hello World" isn't very interesting, but as you create programs that do different actions based on real world input, Serial.print becomes much more useful.

Connecting A Pushbutton

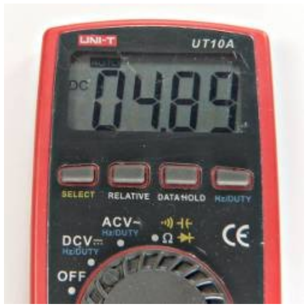
Pushbuttons are simple but very useful input. To connect a pushbutton, you will also need to connect a pullup resistor.



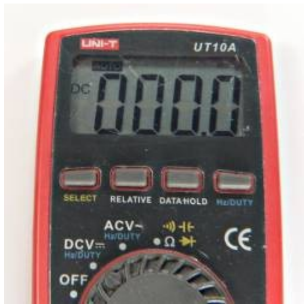
When you press the pushbutton, it connects to ground. When you release, the resistor provides 5 volts on the pushbutton pin. Without the resistor, the voltage could remain at zero. It is called a "pullup" because it serves to pull the voltage back up when you are not pressing the button.

You can verify the circuit works by connecting a voltmeter. The red lead connects to the pushbutton pin and the black lead connects to ground. Here is what you should see on the meter.

Button Not Pressed

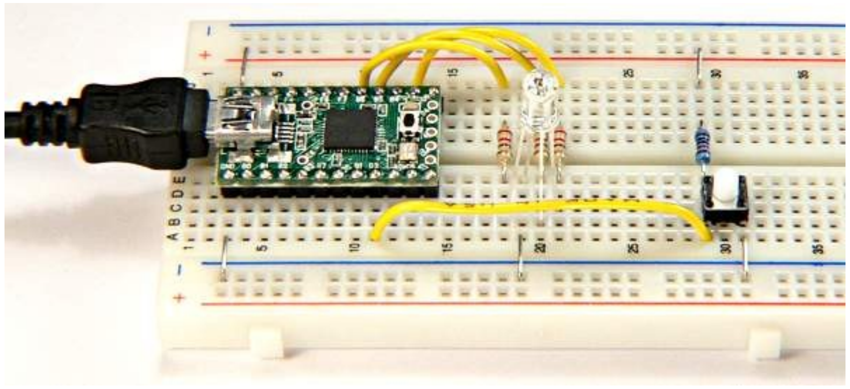


Button Pressed



It may seem strange to have zero when the button is pressed and have 5 volts when it is released. This is called "active low", because the signal is low when the condition is happening. Active low signals are very common practice in electronics, especially for pushbuttons.

When you are confident the pushbutton circuit works, just connect it to any unused pin. In this photo, it is connected to pin 7.



Using digitalRead

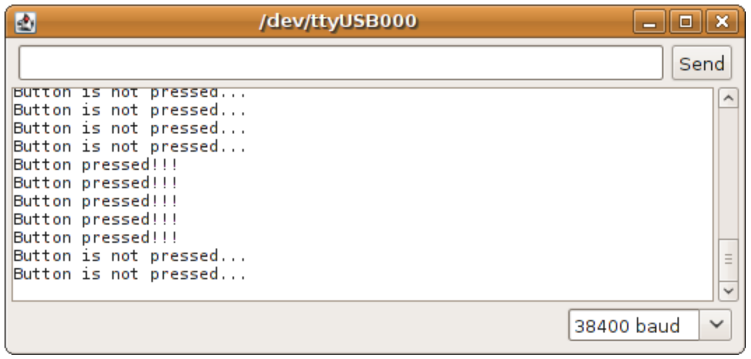
To read the pushbutton, you must first use pinMode to configure the pin to work as an input. Then you can use digitalRead to actually read the pin. Here is an example.

```
void setup()   {
  Serial.begin(38400);
  pinMode(7, INPUT);
}

void loop()
{
  if (digitalRead(7) == HIGH) {
    Serial.println("Button is not pressed...");
  } else {
    Serial.println("Button pressed!!!");
  }
  delay(250);
}
```

In this example, the "if" statement is used, together with "else". The condition after "if" is tested, and either the code in the first or the second set of curly braces is run.

Using digitalRead(7) will result in either HIGH or LOW. This code compares digitalRead(7) to HIGH using the "==" comparison. A subtle but very important detail is the double equal. Using only a single equal attempts to write or assign a value, which would be silly since you can't change what digitalRead(7) found. A double equal means to compare 2 numbers, rather than set the first equal to the second.

When you run the program and use the Serial Monitor, you will see messages that change when you press the pushbutton.

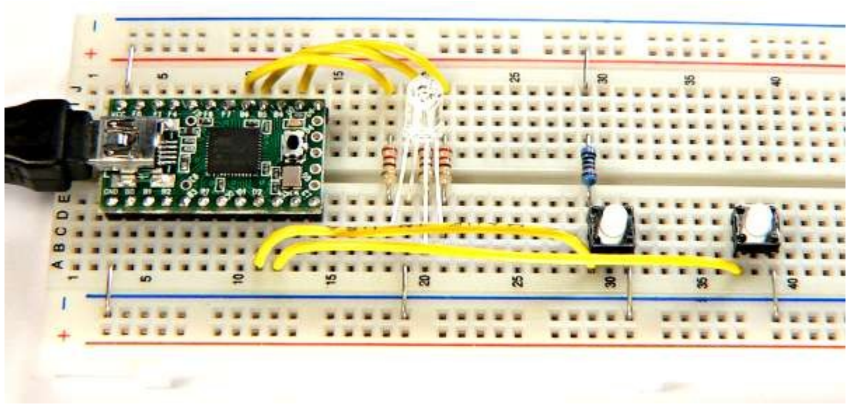


```
Button is not pressed...
Button is not pressed...
Button is not pressed...
Button is not pressed...
Button pressed!!!
Button pressed!!!
Button pressed!!!
Button pressed!!!
Button pressed!!!
Button is not pressed...
Button is not pressed...
```

Built In Pullup Resistor

Use of pullup resistors is so common that the AVR processor on the Teensy board provides a pullup resistor on every pin, built inside the chip itself.

You can connect a second pushbutton by just wiring one side to ground and the other side to a pin (pin 8 in this example). One disadvantage of the built-in pullup is you can not test the pushbutton with the voltmeter before wiring it to your Teensy.



You can activate the pullup resistor by using INPUT_PULLUP with pinMode in your setup function.

```
void setup()   {
  Serial.begin(38400);
  pinMode(8, INPUT_PULLUP);
}
```

Usually when using the pullup resistor, you will turn it on with INPUT_PULLUP and leave it on. However, you can turn it on and off. When the pin is in input mode, you can turn the pullup resistor on and off by using digitalWrite. That isn't very intuitive, writing to a pin which is input mode, but that is how it works. INPUT_PULLUP is a Teensy extension. On regular Arduino boards, digitalWrite the only way to access the pullup resistor.

Pushbuttons To Control LED Colors

You can use digitalRead to make your programs respond to pushbutton input. Here is the Red/Green color fade example from Tutorial 2, but with another variable added so it has 2 modes, fading from Red/Green and Red/Blue. The pushbuttons on pins 7 and 8 are checked each time the loop function runs and the mode is changed if the buttons are pressed.

(You can open these examples and copy-and-paste from your browser into Arduino)

```
int redPin =  12;
int greenPin =  15;
int bluePin =  14;

void setup()   {
  Serial.begin(38400);
  pinMode(7, INPUT);
  pinMode(8, INPUT_PULLUP);
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

int redIntensity = 0;
int mode = 0;

void loop()
{
  // set all 3 pins to the desired intensity
  analogWrite(redPin, redIntensity);
  if (mode == 0) {
    // in mode zero, fade from red to green
    analogWrite(greenPin, 255 - redIntensity);
    analogWrite(bluePin, 0);
  } else {
    // in mode one, fade from red to blue
    analogWrite(greenPin, 0);
    analogWrite(bluePin, 255 - redIntensity);
  }

  if (digitalRead(7) == LOW) {
    // use mode zero when the first button is pressed
    mode = 0;
    Serial.println("mode 0");
  }
  if (digitalRead(8) == LOW) {
    // use mode one when the first button is pressed
    mode = 1;
    Serial.println("mode 1");
  }

  // remain at this color, but not for very long
  delay(10);

  // increase the red
  redIntensity = redIntensity + 1;

  // since 255 is the maximum, set it back to 0
  // when it increments beyond 255
  if (redIntensity > 255) {
    redIntensity = 0;
  }
}
```

With the ability to change output and respond to input pins, you can create very useful projects.

As you create larger programs, use of comments will greatly help you manage how it functions. If your program does not appear to do what you want, using Serial.print and the Serial Monitor can help you learn what it is actually doing.

Can you modify this program to produce other color effects that respond to the pushbuttons?

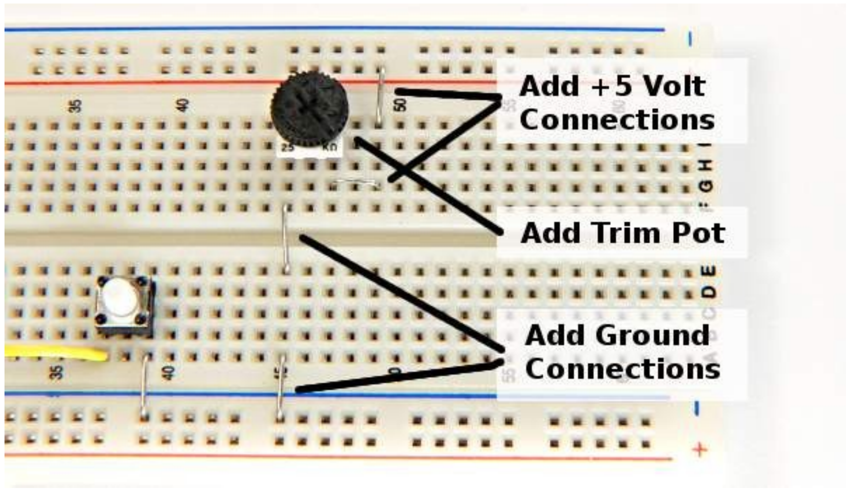
The pushbuttons give you simple on/off intput. In Tutorial 4 you will connect and read analog signals that can vary rather than be limited to only on and off.

Tutorial 4: Analog Input

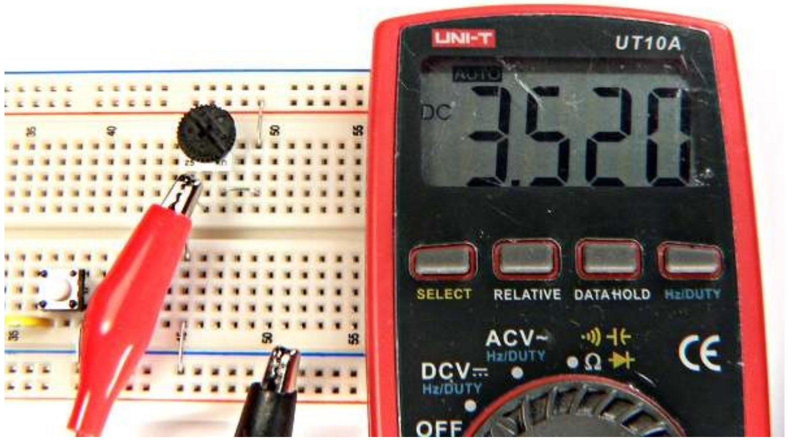
Many important types of signals vary continuously, such a temperature. You can measure such signals with the analog inputs.

Connecting a Potentiometer

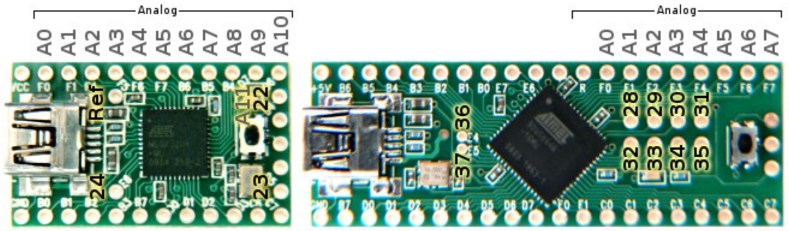
A potentiometer (or "pot") is a resistor with a third pin attached to a mechanical adjustment, so it can slide to any position. You can connect the outside 2 pins to +5 volt power and ground, and when you turn the pot, the middle pin will have a voltage that corresponds to the position.



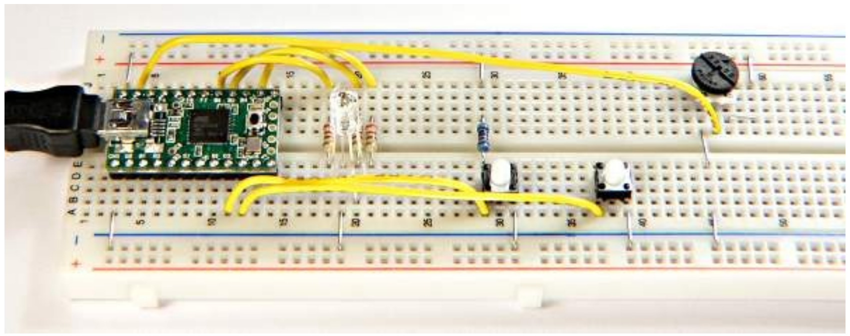
You can check the voltage from the pot by connecting a voltmeter, with the red lead to the center pin on the pot and the black lead to ground. If the leads wiggle loose, you can bend a longer U-shaped wire and plug it into 2 holes, which will be much stronger when you clip a lead onto it.



The Teensy 2.0 has 12 analog input pins (11 are on the edge you can use with a breadboard), and the Teensy++ has 8. The original Teensy 1.0 did not have any analog inputs. Here are the analog capable pins.



For this example, connect the pot's voltage to analog pin 0. When used as analog pins, the Arduino software uses a separate set of zero-based numbers, so pin 0 (used with pinMode, digitalWrite, analogWrite, and digitalRead) is different than analog pin 0.



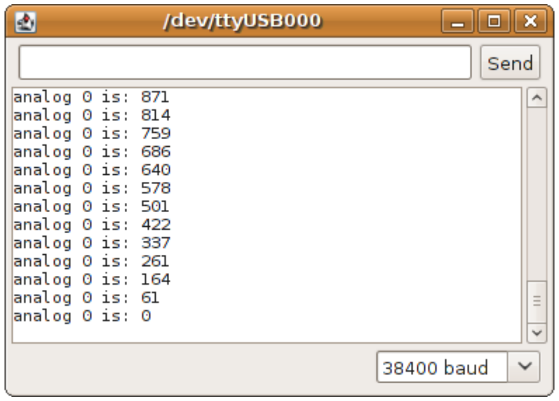
Using analogRead

You can test the analog input with this very simple program.

```
void setup()
{
  Serial.begin(38400);
}

int val;

void loop()
{
  val = analogRead(0);
  Serial.print("analog 0 is: ");
  Serial.println(val);
  delay(250);
}
```

The "analogRead" function reads the voltage on an analog pin, which is assigned to a variable. Text and the variable are printed to the Serial Monitor using Serial.print and Serial.println.

Here is what you should see when slowly turning the knob counter clockwise.



```
analog 0 is: 871
analog 0 is: 814
analog 0 is: 759
analog 0 is: 686
analog 0 is: 640
analog 0 is: 578
analog 0 is: 501
analog 0 is: 422
analog 0 is: 337
analog 0 is: 261
analog 0 is: 164
analog 0 is: 61
analog 0 is: 0
```

When the analog input is at 5 volts (or whatever voltage the chip is actually operating at, if not exactly 5 volts), you should get 1023 and when it is at 0 volts, you should get 0.

User Controlled LED Color

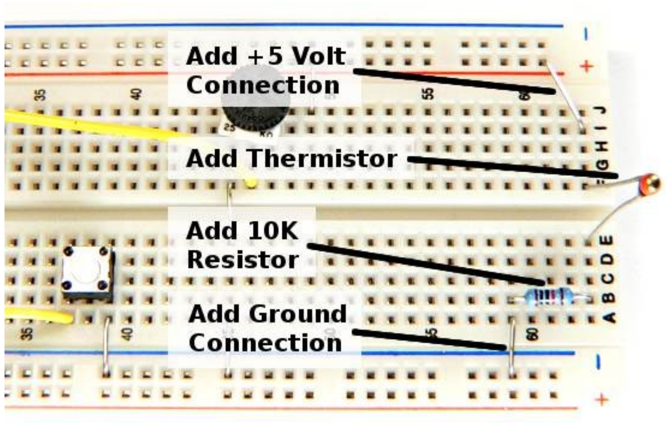
You can use the pot as user input. Here is an example that uses the pot to control the LED color.

```
int redPin =  12;
int greenPin =  15;
int bluePin =  14;

void setup()   {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

int redIntensity = 0;

void loop()
{
  // read the pot position
  redIntensity = analogRead(0) / 4;

  // set all 3 pins to the desired intensity
  analogWrite(redPin, redIntensity);
  analogWrite(greenPin, 255 - redIntensity);
  analogWrite(bluePin, 0);

  // remain at this color, but not for very long
  delay(10);
}
```

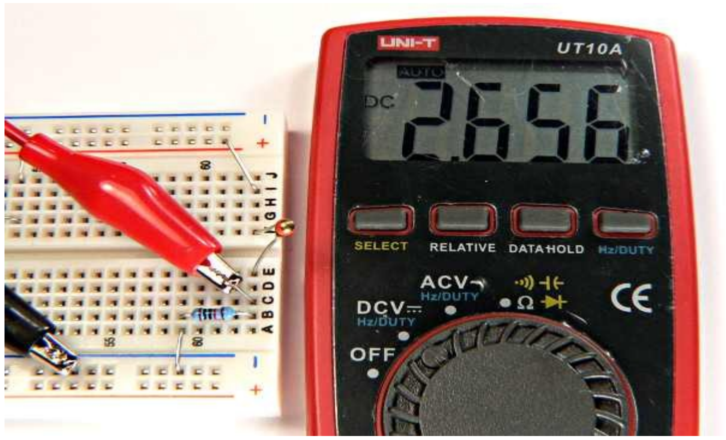
This is similar to the LED color fade from Tutorial 2. However, instead of changing the red intensity automatically, analogRead(0) is used to measure the pot's position. Because analogRead(0) returns a number between 0 to 1023, it is divided by 4 so it will be within the 0 to 255 ranges that analogWrite requires.

Connecting a Temperature Sensor

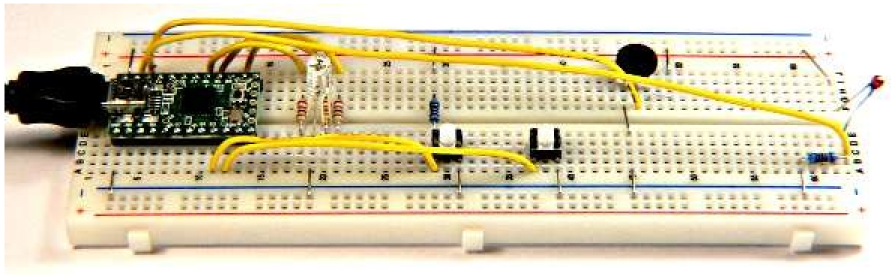
A thermistor is a resistor that changes with temperature. You can connect it together with another resistor that does not change, to create a voltage.



In this example, the thermistor is a Vishay NTCLE100E3103 which is 10K at 25 degrees Celsius and decreases in resistance as the temperature increases. At room temperature, the voltage should be slightly above 2.5 volts. As always, you can check the circuit before connecting it to your Teensy. If you touch the thermistor with your finger, you should see the voltage change.



When you are confident the thermistor is working, connect it to an analog input pin. In this example, it is connected to analog input 1.



Simple Temperature Programs

Just like the pot, you can use a simple program to read the analog input and print the value. As the temperature changes, you should see the number change in the Serial Monitor.

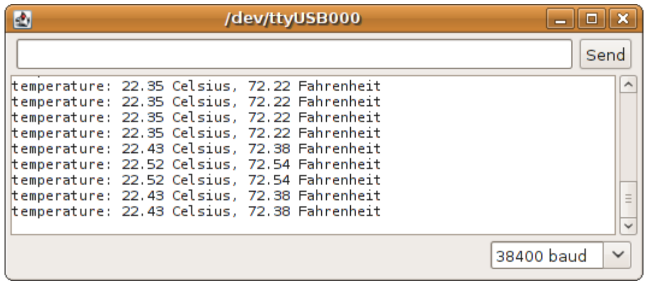
```
void setup()
{
  Serial.begin(38400);
}

int val;

void loop()
{
  val = analogRead(1);
  Serial.print("analog 1 is: ");
  Serial.println(val);
  delay(1000);
}
```

Displaying the number from analogRead isn't very meaningful. Here is an example that converts the number to meaningful temperature numbers.

```
void setup()
{
  Serial.begin(38400);
}

float code;
float celsius;
float fahrenheit;

void loop()
{
  code = analogRead(1);
  celsius = 25 + (code - 512) / 11.3;
  fahrenheit = celsius * 1.8 + 32;
  Serial.print("temperature: ");
  Serial.print(celsius);
  Serial.print(" Celsius, ");
  Serial.print(fahrenheit);
  Serial.println(" Fahrenheit");
  delay(1000);
}
```

In previous examples, variables were created with "int" type, which limits the variable to holding integers. Since temperatures are real numbers with digits past the decimal point, you need to use "float" type variables.

When you run this program, you should see this in the Serial Monitor window.



```
temperature: 22.35 Celsius, 72.22 Fahrenheit
temperature: 22.35 Celsius, 72.22 Fahrenheit
temperature: 22.35 Celsius, 72.22 Fahrenheit
temperature: 22.35 Celsius, 72.22 Fahrenheit
temperature: 22.43 Celsius, 72.38 Fahrenheit
temperature: 22.52 Celsius, 72.54 Fahrenheit
temperature: 22.52 Celsius, 72.54 Fahrenheit
temperature: 22.43 Celsius, 72.38 Fahrenheit
temperature: 22.43 Celsius, 72.38 Fahrenheit
```

More Accurate Temperature Calculation

You might wonder about the equation in the previous example which converts the code from analogRead into the Celsius temperature. The truth is, it is only accurate close to room temperature.

This table shows the voltage and code that will be read at various temperatures. The last column shows the average number of codes per degree in each range. Near 25 degrees, the change is about 11.3 codes per degree. The equation above simply computes the difference between the reading and 25 degrees, then divides by 11.3 codes per degree. Obviously, for temperatures substantially different from 25 degrees, this becomes very inaccurate.

Temp, C	Ohms	Voltage	Code	Code/Deg
-40	332094	0.15	30	
-35	239900	0.20	41	2.21
-30	175200	0.27	55	2.86
-25	129287	0.36	73	3.64
-20	96358	0.47	96	4.55
-15	72500	0.61	124	5.56
-10	55046	0.77	157	6.65
-5	42157	0.96	196	7.77
0	32554	1.17	240	8.85
5	25339	1.41	289	9.82
10	19872	1.67	342	10.60
15	15698	1.95	398	11.12
20	12488	2.22	455	11.36
25	10000	2.50	512	11.32
30	8059	2.77	566	11.00
35	6535	3.02	619	10.44
40	5330	3.26	667	9.73
45	4372	3.48	712	8.90

50	3605	3.68	752	8.03
55	2989	3.85	788	7.13
60	2490	4.00	819	6.29
65	2084	4.14	847	5.50
70	1753	4.25	870	4.77
75	1481	4.36	891	4.12
80	1256	4.44	909	3.56
85	1070	4.52	924	3.05
90	915.4	4.58	937	2.62
95	786	4.64	948	2.25
100	677.3	4.68	958	1.93
105	585.7	4.72	966	1.66
110	508.3	4.76	974	1.42
115	442.6	4.79	980	1.22
120	386.6	4.81	985	1.06
125	338.7	4.84	989	0.91
130	297.7	4.86	993	0.79
135	262.4	4.87	997	0.68
140	231.9	4.89	1000	0.59
145	205.5	4.90	1002	0.52
150	182.6	4.91	1005	0.45

The temperature and ohms data for this sensor can be found in the sensor's datasheet on page 84 (10th page of the PDF file).

In each 5 degree range, the equation is actually different. A simple approach is to simply use a the correct equation in each range. Here is one possible way.

```
void setup()
{
  Serial.begin(38400);
}
```

```
int code;
float celsius;
float fahrenheit;

void loop()
{
  code = analogRead(1);
  if (code <= 289) {
    celsius = 5 + (code - 289) / 9.82;
  }
  if (code > 289 && code <= 342) {
    celsius = 10 + (code - 342) / 10.60;
  }
  if (code > 342 && code <= 398) {
    celsius = 15 + (code - 398) / 11.12;
  }
  if (code > 398 && code <= 455) {
    celsius = 20 + (code - 455) / 11.36;
  }
  if (code > 455 && code <= 512) {
    celsius = 25 + (code - 512) / 11.32;
  }
  if (code > 512 && code <= 566) {
    celsius = 30 + (code - 566) / 11.00;
  }
  if (code > 566 && code <= 619) {
    celsius = 35 + (code - 619) / 10.44;
  }
  if (code > 619 && code <= 667) {
    celsius = 40 + (code - 667) / 9.73;
  }
  if (code > 667) {
    celsius = 45 + (code - 712) / 8.90;
  }
  fahrenheit = celsius * 1.8 + 32;
  Serial.print("temperature: ");
  Serial.print(celsius);
  Serial.print(" Celsius, ");
  Serial.print(fahrenheit);
  Serial.println(" Fahrenheit");
  delay(1000);
}
```

There are many other very sophisticated ways to converting measurement to accurate temperature measurements. The sensor data itself is subject to manufacturing tolerance. For very accurate measurements, you can calibrate the particular sensor you are using by carefully measuring it at precise temperatures, instead of depending on the numbers from the table. But for many simple uses, even the simple equation is often good enough.

Of course, you can use the temperature data in various ways, such as turning on an LED, or perhaps an air conditioner!