Fading a LED

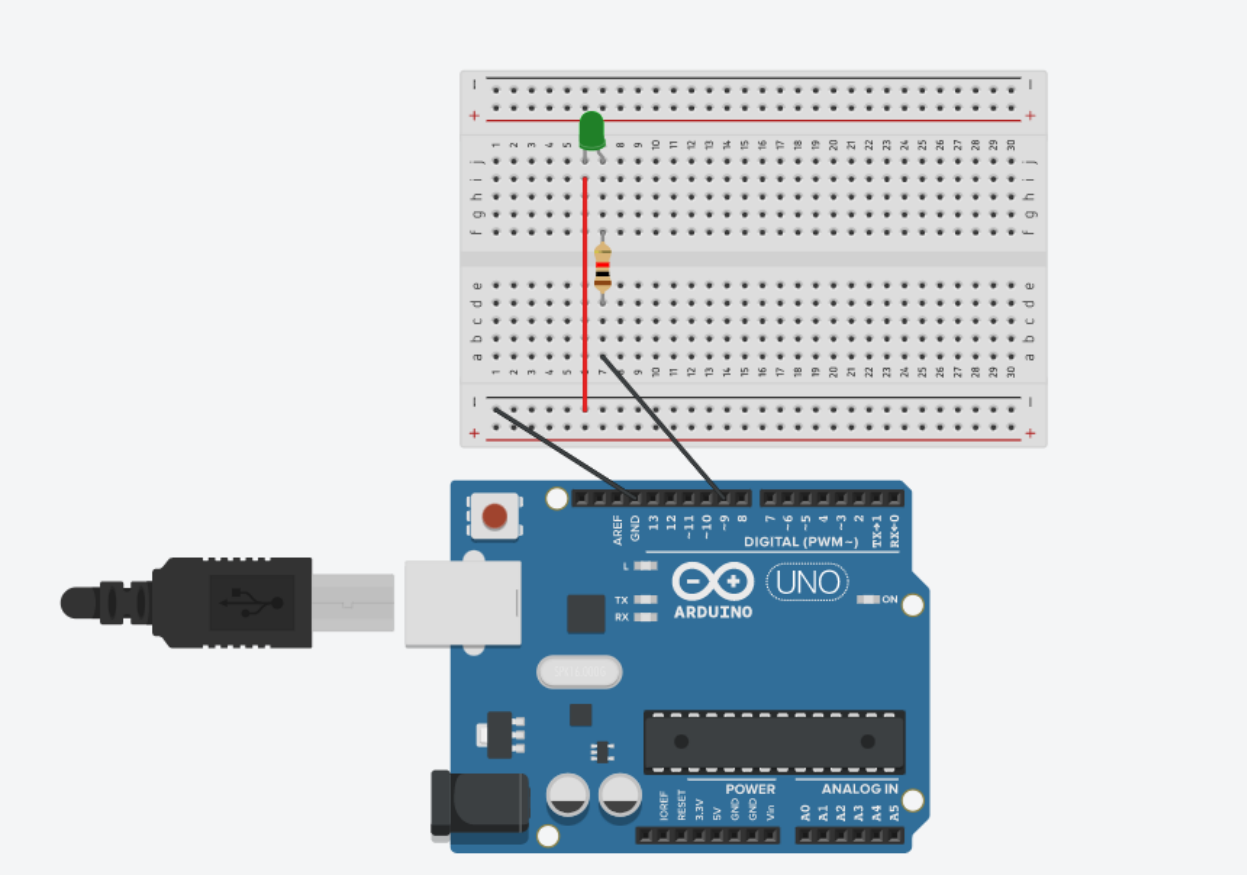
This example demonstrates the use of the analogWrite() function in fading an LED off and on. AnalogWrite uses pulse width modulation (PWM), turning a digital pin on and off very quickly with different ratio between on and off, to create a fading effect.

Hardware Required

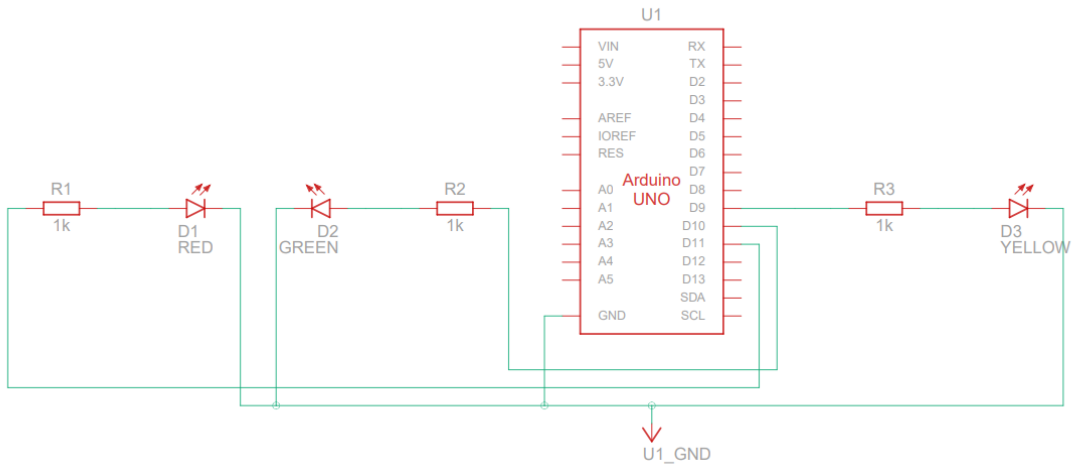
- Arduino board
- LED
- 220 ohm resistor
- hook-up wires
- breadboard

Circuit

Connect the **anode** (the longer, positive leg) of your LED to digital output pin 9 on your board through a 220 ohm resistor. Connect the **cathode** (the shorter, negative leg) directly to ground.


DIGITAL (PWM~)
UNO
ARDUINO
POWER
ANALOG IN

Schematic


U1
VIN
5V
3.3V
AREF
IOREF
RES
A0
A1
A2
A3
A4
A5
GND
RX
TX
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
SDA
SCL
Arduino
UNO
R1
1k
D1
RED
D2
GREEN
R2
1k
R3
1k
D3
YELLOW
U1_GND

Code

After declaring pin 9 to be your led Pin, there is nothing to do in the setup() function of your code.

The analogWrite() function that you will be using in the main loop of your code requires two arguments: One telling the function which pin to write to, and one indicating what PWM value to write. Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between the full Vcc of the board (e.g., 5 V on UNO, 3.3 V on a MKR board) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and Vcc controlling the brightness of the LED.

In order to fade your LED off and on, gradually increase your PWM value from 0 (all the way off) to 255 (all the way on), and then back to 0 once again to complete the cycle. In the source code, the PWM value is set using a variable called brightness. Each time through the loop, it increases by the value of the variable fade Amount.

If brightness is at either extreme of its value (either 0 or 255), then fade Amount is changed to its negative. In other words, if fade Amount is 5, then it is set to -5. If it's -5, then it's set to 5. The next time through the loop, this change causes brightness to change direction as well.

analogWrite() can change the PWM value very fast, so the delay at the end of the sketch controls the speed of the fade. Try changing the value of the delay and see how it changes the fading effect.

COMPLETE CODE:

```
int led = 9;          // the PWM pin the LED is attached to
int brightness = 0;  // how bright the LED is
int fadeAmount = 5;  // how many points to fade the LED by
void setup()
```

```
{
  pinMode(led, OUTPUT);
}
void loop()
{
analogWrite(led, brightness);
  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;
  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255)
  {
    fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

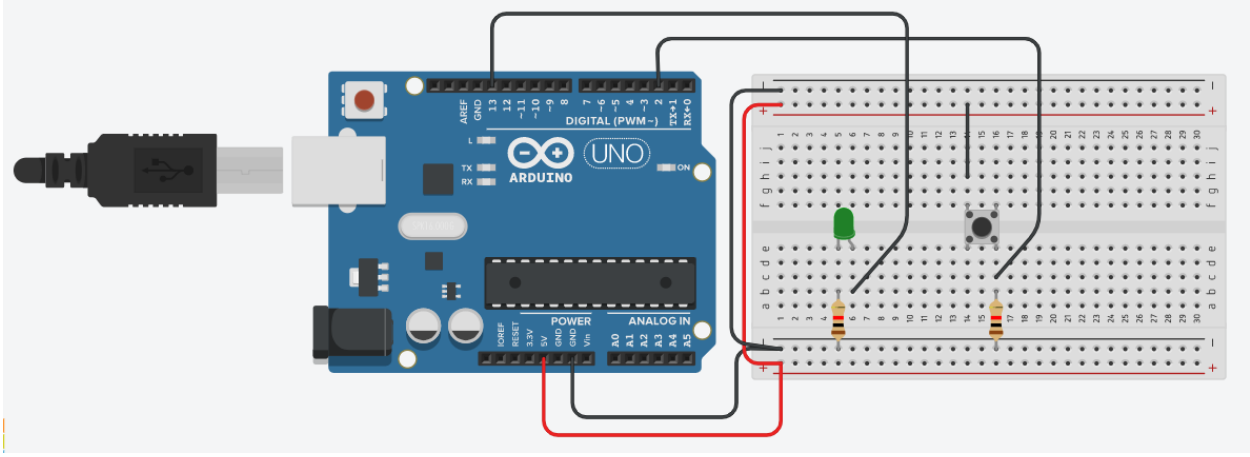
How to Wire and Program a Button

Pushbuttons or switches connect two points in a circuit when you press them. This example turns on the built-in LED on pin 13 when you press the button.

Hardware

- Arduino Board
- Momentary button or Switch

- 10K ohm resistor
- hook-up wires
- breadboard



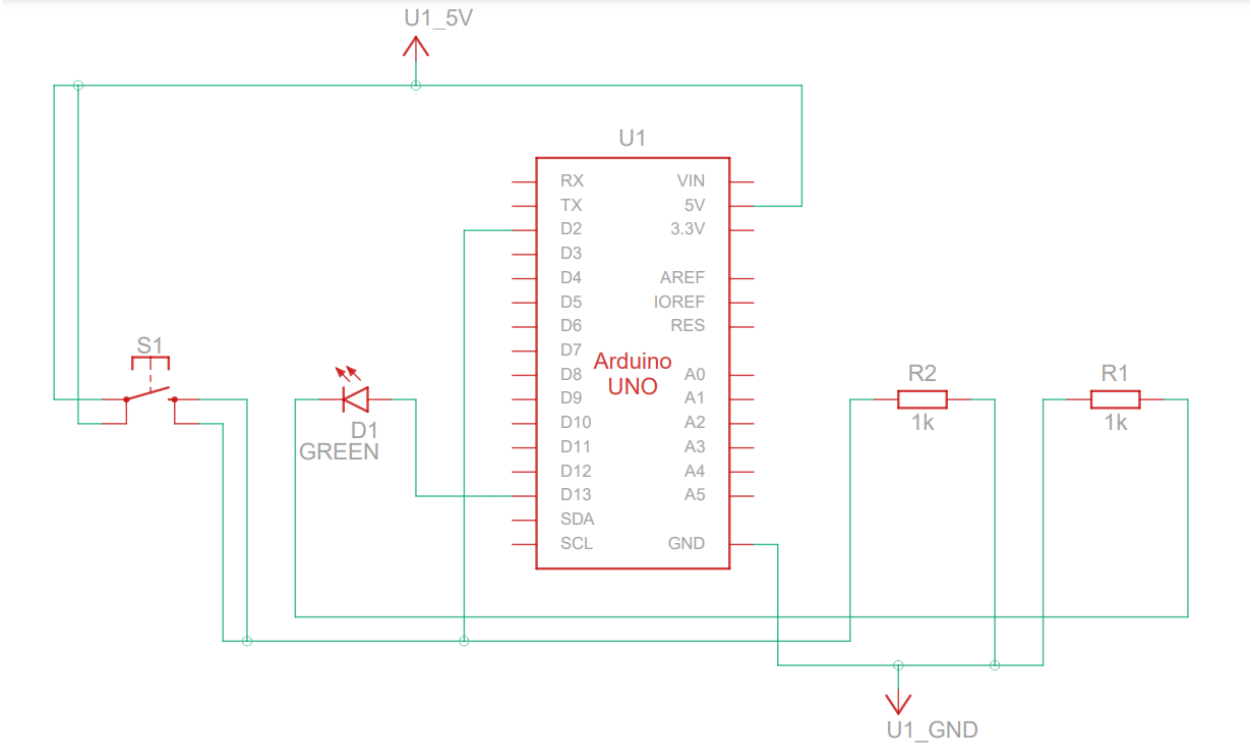
Connect three wires to the board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10K ohm) to ground. The other leg of the button connects to the 5 volt supply.

When the pushbutton is open (unpressed) there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and we read a LOW. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5 volts, so that we read a HIGH.

You can also wire this circuit the opposite way, with a pullup resistor keeping the input HIGH, and going LOW when the button is pressed. If so, the behavior of the sketch will be reversed, with the LED normally on and turning off when you press the button.

If you disconnect the digital I/O pin from everything, the LED may blink erratically. This is because the input is "floating" - that is, it will randomly return either HIGH or LOW. That's why you need a pull-up or pull-down resistor in the circuit.

Schematic



COMPLETE CODE:

```
// C++ code
//
int buttonState=0;
void setup()
{
  pinMode(2, INPUT);
  pinMode(13, INPUT);
}
```

```
void loop()
{
  //read the stae of the push button
  buttonState=digitalRead(2);
  //check if pushbutton is pressed.
  if(buttonState==HIGH)
  {
    digitalWrite(13,HIGH);
  }
  else
  {
   digitalWrite(13,LOW);
  }
  delay(10); // Wait for 10 millisecond(s)
}
```