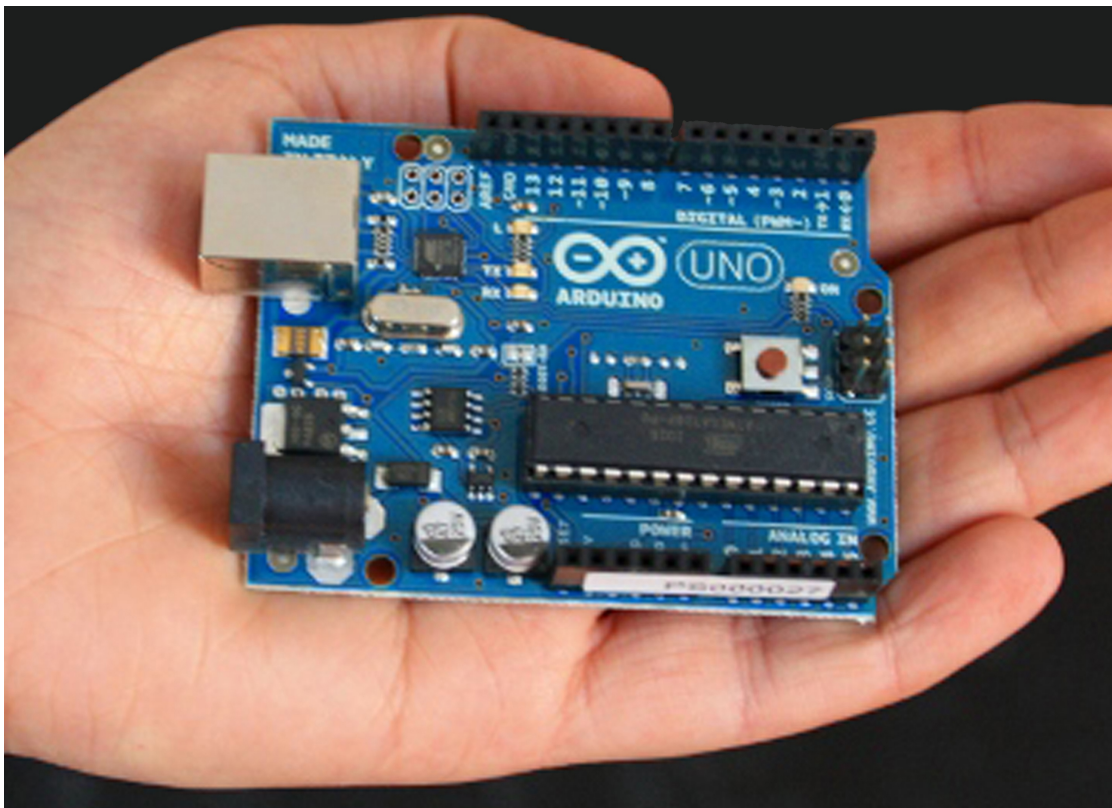


UCHS Computer Science 1

Arduino

Name: _____



Introduction

Some of the key features of the Arduino Uno include:

1. An open source design. The advantage of it being open source is that it has a large community of people using and troubleshooting it. This makes it easy to find someone to help you debug your projects.
2. An easy USB interface . The chip on the board plugs straight into your USB port and registers on your computer as a virtual serial port. This allows you to interface with it as through it were a serial device. The benefit of this setup is that serial communication is an extremely easy (and time-tested) protocol, and USB makes connecting it to modern computers really convenient.
3. Very convenient power management and built-in voltage regulation. You can connect an external power source of up to 12v and it will regulate it to both 5v and 3.3v. It also can be powered directly off of a USB port without any external power.
4. An easy-to-find, and dirt cheap, microcontroller "brain." The ATmega328 chip retails for about \$2.88 on Digikey. It has countless number of nice hardware features like timers, PWM pins, external and internal interrupts, and multiple sleep modes. Check out the official datasheet for more details. A 16mhz clock. This makes it not the speediest microcontroller around, but fast enough for most applications.
5. 32 KB of flash memory for storing your code.
6. 13 digital pins and 6 analog pins. These pins allow you to connect external hardware to your Arduino. These pins are key for extending the computing capability of the Arduino into the real world. Simply plug your devices and sensors into the sockets that correspond to each of these pins and you are good to go.
7. An ICSP connector for bypassing the USB port and interfacing the Arduino directly as a serial device. This port is necessary to re-bootload your chip if it corrupts and can no longer talk to your computer.
8. An on-board LED attached to digital pin 13 for fast an easy debugging of code.
9. And last, but not least, a button to reset the program on the chip.

Getting Familiar with the Uno

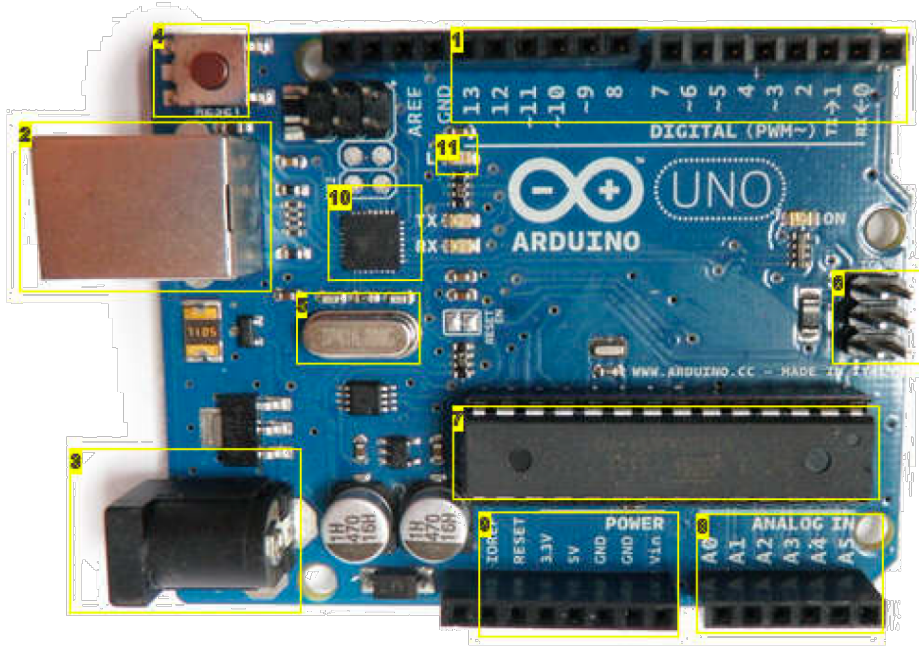


Image Notes

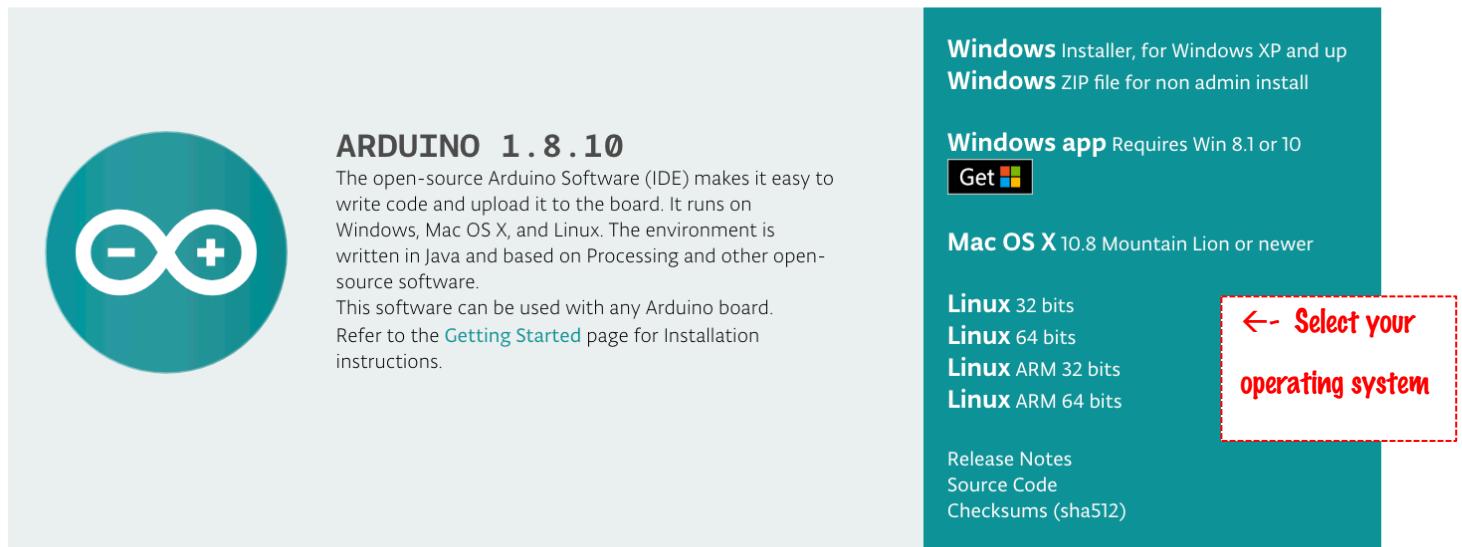
1. Digital Pins
2. USB port
3. External power socket
4. Reset button
5. 16MHZ crystal
6. ICSP - in circuit serial programmer
7. ATMEGA328
8. Analog in
9. Power management sockets
10. virtual serial port chip
11. Status LED connected to pin 13

Setup

Before you can start doing anything with the Arduino, you need to download and install the Arduino IDE (integrated development environment). The Arduino Programmer is based on the Processing IDE and uses a variation of the C and C++ programming languages.

Step 1: Download Software: <https://www.arduino.cc/en/Main/Software>

Download the Arduino IDE



ARDUINO 1.8.10

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

This software can be used with any Arduino board. Refer to the [Getting Started](#) page for Installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10
Get

Mac OS X 10.8 Mountain Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

Release Notes
Source Code
Checksums (sha512)

←- Select your operating system

- Skip the donate and just download now. This will download a zip file to your computer.
 - Create a folder for Arduino in your CS1 file folder on your computer, and save this file there.
 - Open the zip file and unzip the program.
 - Create a shortcut on your computer.
-

Step 2: Connect the Arduino to your computer

- Connect the Arduino to your computer's USB port.
 - Please note that although the Arduino plugs into your computer, it is not a true USB device. The board has a special chip that allows it to show up on your computer as a virtual serial port when it is plugged into a USB port. This is why it is important to plug the board in. When the board is not plugged in, the virtual serial port that the Arduino operates upon will not be present (since all of the information about it lives on the Arduino board).
 - It is also good to know that every single Arduino has a unique virtual serial port address. This means that every time you plug in a different Arduino board into your computer, you will need to reconfigure the serial port that is in use.
 - The Arduino Uno requires a male USB A to male USB B cable .
-

Step 3: Adjust the Settings

- To set the board, go to the following:

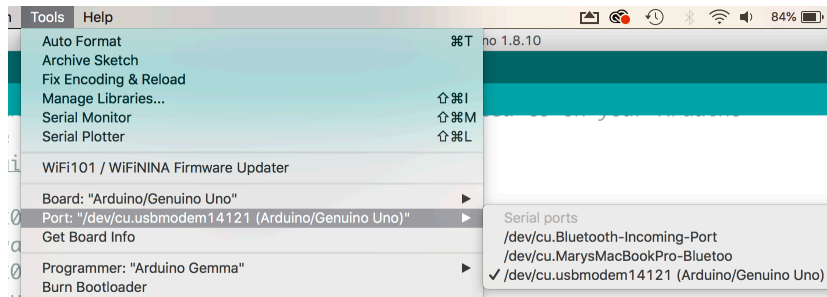
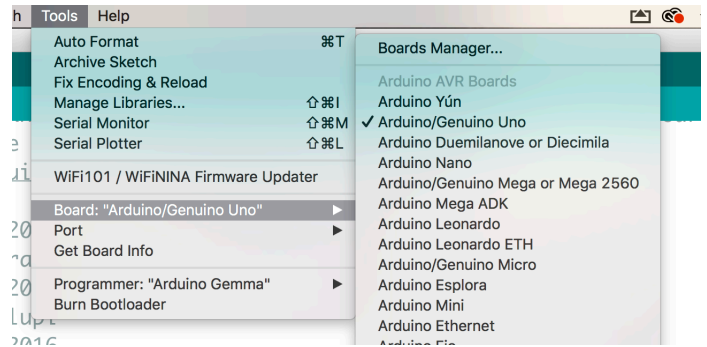
Tools --> Boards

- Select the version of board that you are using.
- To set the serial port, go to the following:

Tools --> Serial Port

- Select the serial port that looks like:

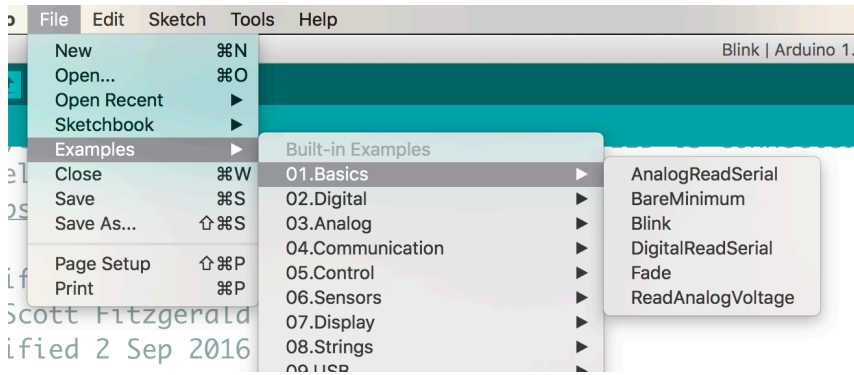
/dev/tty.usbmodem [random numbers]



#1 – Test Arduino

The first project is one of the most basic and simple circuits you can create with Arduino. This project will test your Arduino by blinking an LED that is connected directly to the board.

- Arduino programs are called sketches. The Arduino programmer comes with a ton of example sketches preloaded. This is great because even if you have never programmed anything in your life, you can load one of these sketches and get the Arduino to do something.
- To get the LED on the Arduino which is tied to digital pin 13, let's load the blink example. The blink example can be found here:
Files --> Examples --> Basics --> Blink
- The blink example basically sets pin D13 as an output and then blinks the test LED on the Arduino board on and off every second.
- Once the blink example is open, it can be installed onto the ATMEGA328 chip by pressing the upload button, which looks like an arrow pointing to the right.
- Notice that the surface mount status LED connected to pin 13 on the Arduino will start to blink. You can change the rate of the blinking by changing the length of the delay and pressing the upload button again.



Writing your Own Sketch

Some important things to keep in mind when writing your own code:

- An Arduino program is called a sketch.
- All code in an Arduino sketch is processed from top to bottom.
- Arduino sketches are typically broken into five parts.
 1. The sketch usually starts with a header that explains what the sketch is doing, and who wrote it.
 2. Next, it usually defines global variables. Often, this is where constant names are given to the different Arduino pins.
 3. After the initial variables are set, the Arduino begins the setup routine. In the setup function, we set initial conditions of variables when necessary, and run any preliminary code that we only want to run once. This is where serial communication is initiated, which is required for running the serial monitor.
 4. From the setup function, we go to the loop routine. This is the main routine of the sketch. This is not only where your main code goes, but it will be executed over and over, so long as the sketch continues to run.
 5. Below the loop routine, there is often other functions listed. These functions are user-defined and only activated when called in the setup and loop routine. When these functions are called, the Arduino processes all the code in the function from top to bottom and then goes back to the next line in the sketch where it left off when the function was called. Functions are good because they allow you to run standard routines - over and over - without having to write the same lines of code over and over. You can simply call upon a function multiple times, and this will free up memory on the chip because the function routine is only written once. It also makes code easier to read.
- All of that said, the only two parts of the sketch which are mandatory are the Setup and Loop routines.
- Code must be written in the Arduino Language, which is roughly based on C.

- ❑ Almost all statements written in the Arduino language must end with a ;
 - ❑ Conditionals (such as if statements and for loops) do not need a ;
 - ❑ Conditionals have their own rules and can be found under "Control Structures" on the Arduino Language page
 - ❑ Variables are storage compartments for numbers. You can pass values into and out of variables. Variables must be defined (stated in the code) before they can be used and need to have a data type associated with it.
-

To Start Programming:

First, open the BareMinimum sketch, which can be found at:

File --> Examples --> 1.Basic --> BareMinimum

The BareMinimum Sketch should look like this:

```
void setup() {  
  // put your setup code here, to run once:  
}  
void loop() {  
  // put your main code here, to run repeatedly:  
}
```

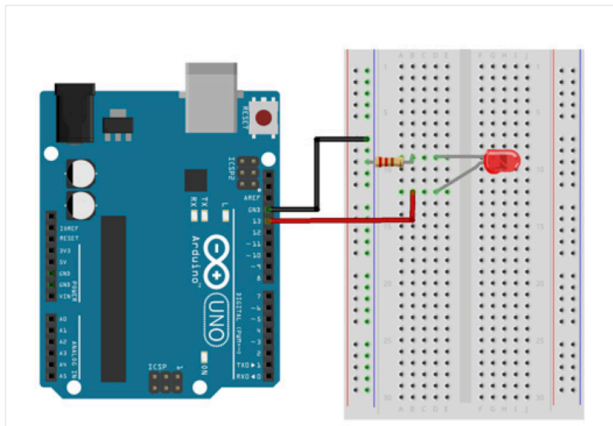
PROJECT #2 BLINK

This project is identical to project #1 except that we will be building it on a breadboard. Once complete, the LED should turn on for a second and then off for a second in a loop.

Parts Needed

- (1) Arduino Uno
- (1) USB A-to-B Cable
- (1) Breadboard – Half Size
- (1) LED 5mm
- (1) 220 Ω Resistor
- (2) Jumper Wires

Project Diagram



```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH); // Turn on the LED
  delay(1000);           // Wait for one second
  digitalWrite(13, LOW); // Turn off the LED
  delay(1000);          // Wait for one second
}
```

**Draw a schematic
of this circuit in
your Arduino
Schematic
Handout**

*REMEMBER to take photos of each
project and challenge!*

REQUIRED CHALLENGES:

1. Add header comments to your sketch
/* Title:
* Description:
* Author:
* Date:
*/
2. Try changing the 1000 in the above delay() functions to different numbers and see how it affects the timing. Smaller values will make the loop run faster. (Why?)
3. Decrease the delay to 10 ms. Can you still see it blink?
4. Find the smallest delay that you can still see a blink. What is this frequency?
5. Modify the code above to resemble a heartbeat.
6. Modify the code above to FLASH SOS repeatedly

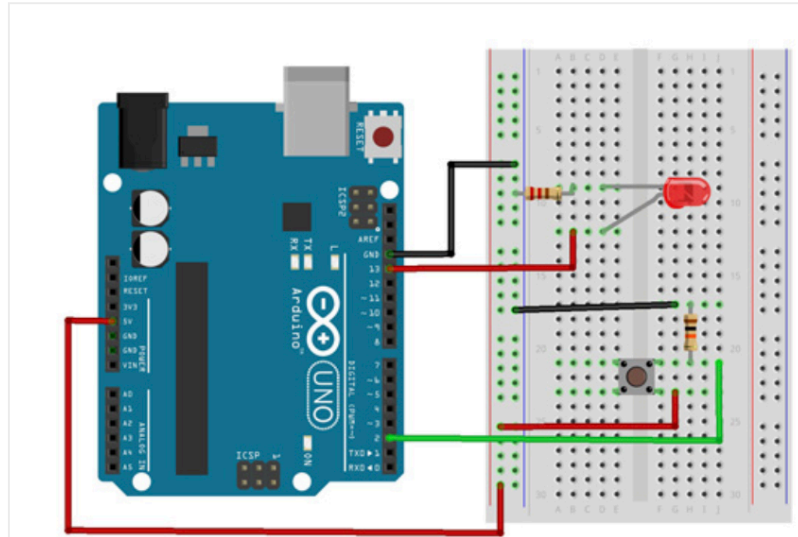
PROJECT #3 Push Button

Project Diagram

Using a push button switch, you will be able to turn on and off an LED.

Parts Needed

- (1) Arduino Uno
- (1) USB A-to-B Cable
- (1) Breadboard – Half Size
- (1) LED 5mm
- (1) 220 Ω Resistor
- (1) 10K Ω Resistor
- (1) Push Button Switch
- (6) Jumper Wires



Push_button

```
1
2 const int buttonPin =2;
3 const int ledPin=8;
4
5 int buttonState = 0;
6
7 void setup() {
8   // put your setup code here, to run once:
9   pinMode(ledPin, OUTPUT);
10  pinMode(buttonPin, INPUT);
11 }
12
13 void loop() {
14   // put your main code here, to run repeatedly:
15   buttonState = digitalRead(buttonPin);
16   if (buttonState == HIGH)
17   {
18     digitalWrite(ledPin, HIGH);
19   }
20   else{
21     digitalWrite(ledPin,LOW);
22   }
23 }
```

Draw a schematic of this circuit in your Arduino Schematic Handout

REMEMBER to take photos of each project and challenge!

REQUIRED CHALLENGE: Answer on schematic sheet,

1. What is the variable button State used for? How is used in this program?

PROJECT #4 Control LEDs

Draw a schematic of this circuit in your Arduino Schematic Handout

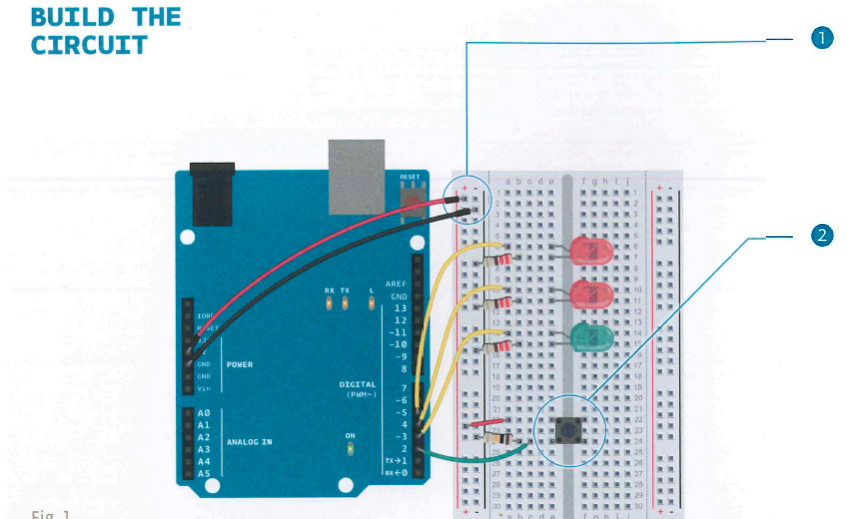
Create a Circuit and write a program as follows:

REMEMBER to take photos of each project and challenge!

- ❑ When the program starts, the green light should be on (SwitchState is LOW),
- ❑ When the user presses and holds the button, the red lights should come on and start flashing
- ❑ When the user releases the button, the green light should turn on and red lights off.

Parts Needed: switch, 3 LEDs (1 green, 2 red), 220 ohm resistor, 10K ohm resistor.

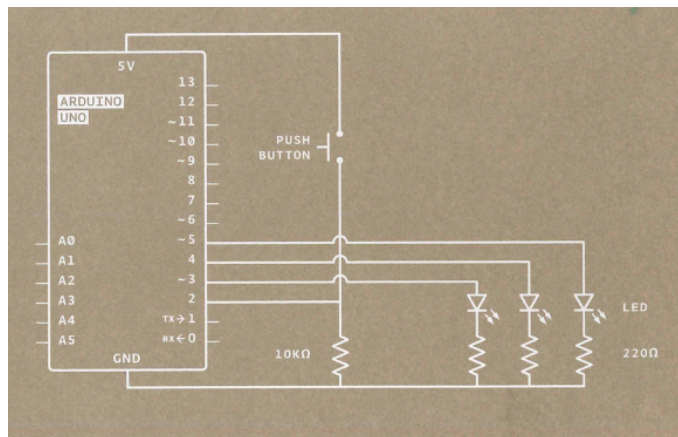
BUILD THE CIRCUIT



1 Wire up your breadboard to the Arduino's 5V and ground connections, just like the previous project. Place the two red LEDs and one green LED on the breadboard. Attach the cathode (short leg) of each LED to ground through a 220-ohm resistor. Connect the anode (long leg) of the green LED to pin 3. Connect the red LEDs' anodes to pins 4 and 5, respectively.

2 Place the switch on the breadboard just as you did in the previous project. Attach one side to power, and the other side to digital pin 2 on the Arduino. You'll also need to add a 10K-ohm resistor from ground to the switch pin that connects to the Arduino. That pull-down resistor connects the pin to ground when the switch is open, so it reads LOW when there is no voltage coming in through the switch.

```
spaceship-interface.ino
1 int switchState = 0;
2
3 void setup() {
4
5   pinMode(3, OUTPUT);
6   pinMode(4, OUTPUT);
7   pinMode(5, OUTPUT);
8   pinMode(2, INPUT);
9
10 }
11
12 void loop() {
13   // put your main code here, to run repeatedly:
14
15   switchState = digitalRead(2);
16
17   if(switchState == LOW){
18     digitalWrite(3, HIGH);
19     digitalWrite(4, LOW);
20     digitalWrite(5, LOW);
21
22   } else {
23     digitalWrite(3, LOW);
24     digitalWrite(4, HIGH);
25     digitalWrite(5, HIGH);
26
27     delay(250);
28
29     digitalWrite(4, HIGH);
30     digitalWrite(5, LOW);
31
32     delay(250);
33   }
34
35 }
```



REQUIRED CHALLENGES:

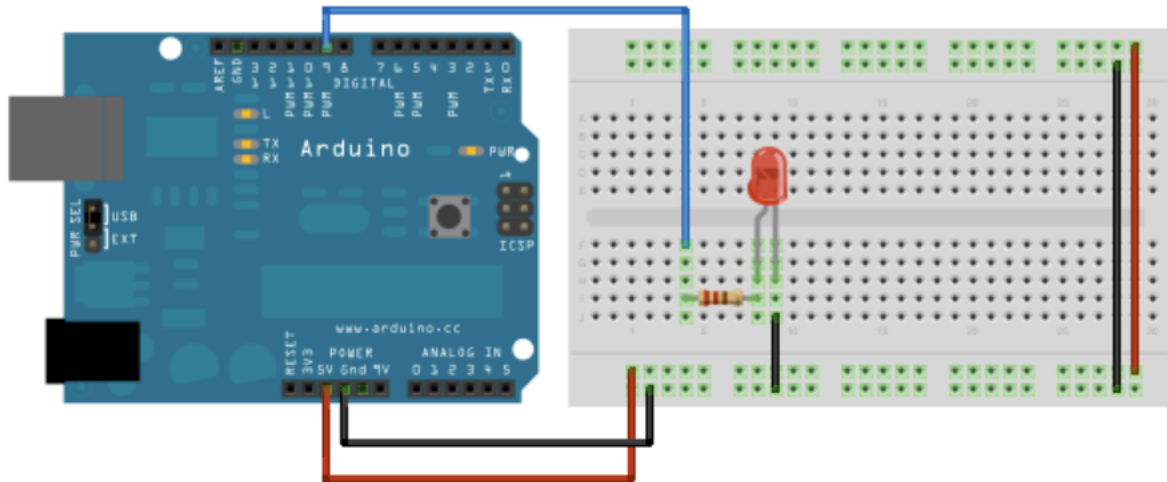
1. Change your program to have the red lights blinking when the program starts, and the green light to turn on when the button is pressed.

PROJECT #5 FADE

How to fade an LED

- ❑ Use pin 9 and the analog function
- ❑ The analogWrite() function uses PWM (Pulse with Modulation), so if you want to change the pin you're using, be sure to use another PWM capable (~) pin.

Parts needed: LED, 220 ohm resistor



```
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

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  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);
}
```

**Draw a schematic
of this circuit in your
Arduino Schematic
Handout**

*REMEMBER to take photos of each
project and challenge!*

REQUIRED CHALLENGES:

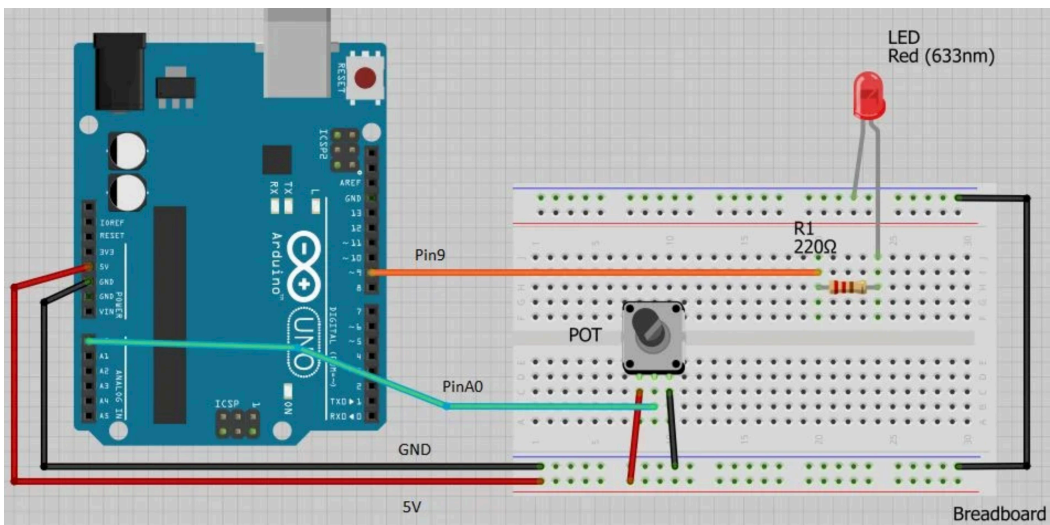
1. Explain what the if statement means. Why is it necessary for the program to run?
2. What happens if you change the delay?

PROJECT #6 Potentiometer

This example shows you how to read an analog input pin, map the result to a range from 0 to 255, use that result to set the pulse width modulation (PWM) of an output pin to dim or brighten an LED and print the values on the serial monitor of the Arduino Software (IDE).

Parts Needed: RED LED, 220 ohm resistor, potentiometer

- ❑ You may have come across the question of how to change the brightness of a LED, without having to keep switching parts. Quite simply, the solution to this issue is a potentiometer.
- ❑ Potentiometers are variable resistors and they function to alter their resistance via a knob or dial. You have probably used one before by adjusting the volume on your stereo or using a light dimmer.
- ❑ Potentiometers have a range of resistance. They can be attuned from zero ohms to whatever maximum resistance that is specific to it. For example, a potentiometer of 10 k Ω can be adjusted from 0 Ω to its maximum of 10 k Ω .
- ❑ In this project, you will learn how to use a potentiometer with and without Arduino board to fade an LED.
- ❑ You will also learn how to use `analogRead()` and `map()` functions.



Draw a schematic of this circuit in your Arduino Schematic Handout

REMEMBER to take photos of each project and challenge!

```
//Constants:
const int ledPin = 9; //pin 9 has PWM funtion
const int potPin = A0; //pin A0 to read analog input

//Variables:
int value; //save analog value

void setup(){
  //Input or output?
  pinMode(ledPin, OUTPUT);
  pinMode(potPin, INPUT); //Optional
}

void loop(){

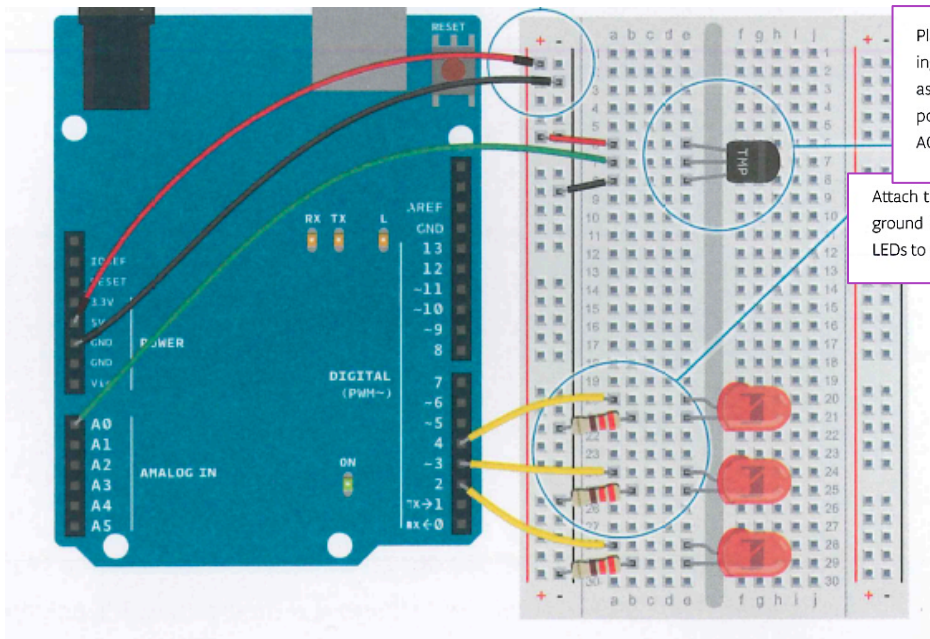
  value = analogRead(potPin); //Read and save analog value from potentiometer
  value = map(value, 0, 1023, 0, 255); //Map value 0-1023 to 0-255 (PWM)
  analogWrite(ledPin, value); //Send PWM value to led
  delay(100); //Small delay
}
```

PROJECT #7 Temperature Sensor

- ❑ How hot are you? Use analog input, you are going to register just how hot you really are!
- ❑ Analog sensors measure things like temperature or light. To do this, you will take advantage of the Arduino's built-in Analog to Digital Converter (ADC). Analog in pins A0-A5 can report back a value between 0-1023, which maps to a range from 0 to 5 volts.
- ❑ You'll be using a temperature sensor to measure how warm your skin is.
- ❑ In the sketch for this project, you will read the sensor's output and use it to turn LEDs on and off, indicating how warm you are.

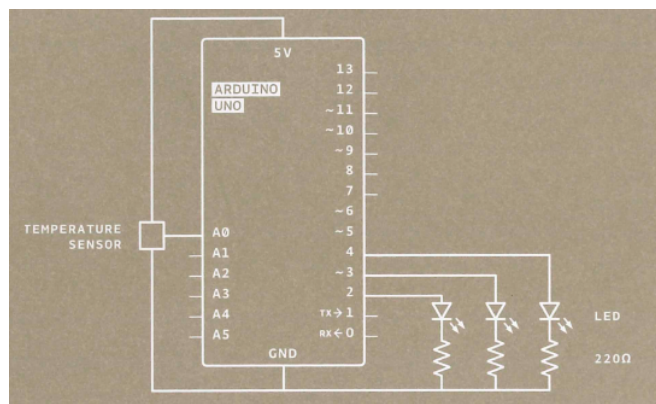
The components

- Arduino UNO
- Breadboard
- Jumper wires
- LEDs
- 220 ohm resistors
- TMP36 temperature sensor



Place the TMP36 on the breadboard with the rounded part facing away from the Arduino (the order of the pins is important!) as shown in Fig. 2. Connect the left pin of the flat facing side to power, and the right pin to ground. Connect the center pin to pin A0 on your Arduino. This is analog input pin 0.

Attach the cathode (short leg) of each of the LEDs you're using to ground through a 220-ohm resistor. Connect the anodes of the LEDs to pins 2 through 4. These will be the indicators for the project.



```

// named constant for the pin the sensor is connected to
const int sensorPin = A0;
// room temperature in Celsius
const float baselineTemp = 20.0;

void setup() {
  // open a serial connection to display values
  Serial.begin(9600);
  // set the LED pins as outputs
  // the for() loop saves some extra coding
  for (int pinNumber = 2; pinNumber < 5; pinNumber++) {
    pinMode(pinNumber, OUTPUT);
    digitalWrite(pinNumber, LOW);
  }
}

void loop() {
  // read the value on AnalogIn pin 0 and store it in a variable
  int sensorVal = analogRead(sensorPin);

  // send the 10-bit sensor value out the serial port
  Serial.print("sensor Value: ");
  Serial.print(sensorVal);

  // convert the ADC reading to voltage
  float voltage = (sensorVal / 1024.0) * 5.0;

  // Send the voltage level out the Serial port
  Serial.print(", Volts: ");
  Serial.print(voltage);

  // convert the voltage to temperature in degrees C
  // the sensor changes 10 mV per degree
  // the datasheet says there's a 500 mV offset
  // ((voltage - 500 mV) times 100)
  Serial.print(", degrees C: ");
  float temperature = (voltage - .5) * 100;
  Serial.println(temperature);

  // if the current temperature is lower than the baseline turn off all LEDs
  if (temperature < baselineTemp + 2) {
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
  } // if the temperature rises 2-4 degrees, turn an LED on
  else if (temperature >= baselineTemp + 2 && temperature < baselineTemp + 4) {
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
  } // if the temperature rises 4-6 degrees, turn a second LED on
  else if (temperature >= baselineTemp + 4 && temperature < baselineTemp + 6) {
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);
  } // if the temperature rises more than 6 degrees, turn all LEDs on
  else if (temperature >= baselineTemp + 6) {
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
  }
  delay(1);
}

```

PROJECT #8 RGB LED and Color Mixer

- ❑ Walk through the "main" colors the RGB LED light can produce
- ❑ Show a rainbow of colors!
- ❑ Learn about functions - Breaking down tasks down into individual functions like this makes your code easier to follow, and it allows parts of your code to be re-used.
- ❑ Write your own color mixing function and add it to the sketch

Components:

3 300 Ω Resistors

RGB LED

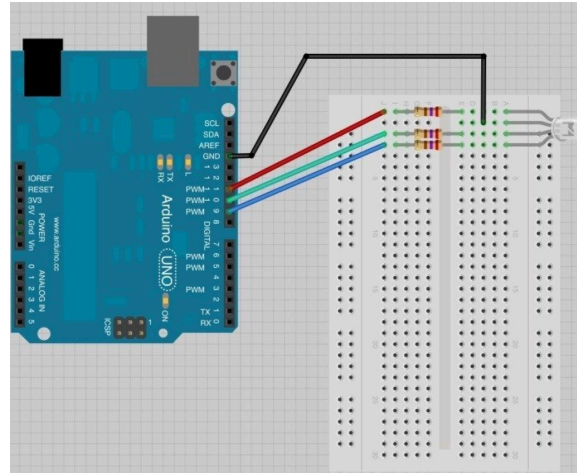
Notice the order of the pins. The RGB LED has 4 legs.

1: RED

2: GROUND

3: GREEN

4: BLUE



You will use the *analogWrite* function of Arduino to control the color of the LED.

At first glance, RGB (Red, Green, Blue) LEDs look just like regular LEDs, however, inside the usual LED package, there are actually three LEDs, one red, one green and yes, one blue. By controlling the brightness of each of the individual LEDs you can mix pretty much any color you want.

We mix colors just like you would mix audio with a 'mixing board' or paint on a palette - by adjusting the brightness of each of the three LEDs. The hard way to do this would be to use different value resistors (or variable resistors) as we played with in lesson 2. That's a lot of work! Fortunately for us, the Arduino has an *analogWrite* function that you can use with pins marked with a ~ to output a variable amount of power to the appropriate LEDs.

Colors

The reason that you can mix any color you like by varying the quantities of red, green and blue light is that your eye has three types of light receptor in it (red, green and blue). Your eye and brain process the amounts of red, green and blue and convert it into a color of the spectrum.

In a way, by using the three LEDs we are playing a trick on the eye. This same idea is used in TVs, where the LCD has red, green and blue color dots next to each other making up each pixel.

If we set the brightness of all three LEDs to be the same, then the overall color of the light will be white. If we turn off the blue LED, so that just the red and green LEDs are the same brightness, then the light will appear yellow.

We can control the brightness of each of the red, green and blue parts of the LED separately, making it possible to mix any color we like.

Black is not so much a color as an absence of light. So the closest we can come to black with our LED is to turn off all three colors.

```

1 /*****
2
3 * Example sketch -- RGB LED
4 *
5 * Make an RGB LED display a rainbow of colors!
6 *
7 * Breaking down tasks down into individual functions like this
8 * makes your code easier to follow, and it allows.
9 * parts of your code to be re-used.
10 *
11 *****/
12 const int RED_PIN = 7;
13 const int GREEN_PIN = 6;
14 const int BLUE_PIN = 5;
15
16 const int DISPLAY_TIME = 1000; // used in mainColors() to determine the
17 // length of time each color is displayed.
18
19 void setup() //Configure the Arduino pins to be outputs to drive the LEDs
20 {
21   pinMode(RED_PIN, OUTPUT);
22   pinMode(GREEN_PIN, OUTPUT);
23   pinMode(BLUE_PIN, OUTPUT);
24 }
25
26 void loop()
27 {
28   // select a function below to test
29   //mainColors(); // Red, Green, Blue, Yellow, Cyan, Purple, White
30   //showSpectrum(); // Gradual fade from Red to Green to Blue to Red
31 }
32
33 /*****
34 * void mainColors()
35 * This function displays the eight "main" colors that the RGB LED
36 * can produce.
37 *****/
38 void mainColors()
39 {
40   // all LEDs off
41   digitalWrite(RED_PIN, LOW);
42   digitalWrite(GREEN_PIN, LOW);
43   digitalWrite(BLUE_PIN, LOW);
44   delay(DISPLAY_TIME);
45
46   // Red
47   digitalWrite(RED_PIN, HIGH);
48   digitalWrite(GREEN_PIN, LOW);
49   digitalWrite(BLUE_PIN, LOW);
50   delay(DISPLAY_TIME);
51
52   // Green
53   digitalWrite(RED_PIN, LOW);
54   digitalWrite(GREEN_PIN, HIGH);
55   digitalWrite(BLUE_PIN, LOW);
56   delay(DISPLAY_TIME);
57
58   // Blue
59   digitalWrite(RED_PIN, LOW);
60   digitalWrite(GREEN_PIN, LOW);
61   digitalWrite(BLUE_PIN, HIGH);
62   delay(DISPLAY_TIME);
63

```

Draw a schematic of this circuit in your Arduino Schematic Handout

*REMEMBER to take photos of each
project and challenge!*


```

64 // Yellow (Red and Green)
65 digitalWrite(REL_PIN, HIGH);
66 digitalWrite(GREEN_PIN, HIGH);
67 digitalWrite(BLUE_PIN, LOW);
68 delay(DISPLAY_TIME);
69
70 // Cyan (Green and Blue)
71 digitalWrite(REL_PIN, LOW);
72 digitalWrite(GREEN_PIN, HIGH);
73 digitalWrite(BLUE_PIN, HIGH);
74 delay(DISPLAY_TIME);
75
76 // Purple (Red and Blue)
77 digitalWrite(REL_PIN, HIGH);
78 digitalWrite(GREEN_PIN, LOW);
79 digitalWrite(BLUE_PIN, HIGH);
80 delay(DISPLAY_TIME);
81
82 // White (turn all the LEDs on)
83 digitalWrite(REL_PIN, HIGH);
84 digitalWrite(GREEN_PIN, HIGH);
85 digitalWrite(BLUE_PIN, HIGH);
86 delay(DISPLAY_TIME);
87 }

88
89 /*****
90 * void showSpectrum()
91 *
92 * Steps through all the colors of the RGB LED, displaying a rainbow.
93 * showSpectrum() calls a function RGB(int color) that translates a number
94 * from 0 to 767 where 0 = all RED, 767 = all RED
95 *
96 *****/
97
98 void showSpectrum()
99 {
100   for (int x = 0; x <= 767; x++)
101   {
102     RGB(x);    // Increment x and call RGB() to progress through colors.
103     delay(10); // Delay for 10 ms (1/100th of a second) - to help the "smoothing"
104   }
105 }
106
107
108

```

MORE

```

109 /*****
110 * void RGB(int color)
111 *
112 * RGB(###) displays a single color on the RGB LED.
113 * Call RGB(###) with the number of a color you want
114 * to display. For example, RGB(0) displays pure RED, RGB(255)
115 * displays pure green.
116 *
117 * This function translates a number between 0 and 767 into a
118 * specific color on the RGB LED. If you have this number count
119 * through the whole range (0 to 767), the LED will smoothly
120 * change color through the entire spectrum.
121 *
122 * The "base" numbers are:
123 * 0 = pure red
124 * 255 = pure green
125 * 511 = pure blue
126 * 767 = pure red (again)
127 *
128 * Numbers between the above colors will create blends. For
129 * example, 640 is midway between 512 (pure blue) and 767
130 * (pure red). It will give you a 50/50 mix of blue and red,
131 * resulting in purple.
132 /*****/
133 void RGB(int color)
134 {
135     int redIntensity;
136     int greenIntensity;
137     int blueIntensity;
138
139     color = constrain(color, 0, 767); // constrain the input value to a range of values from 0 to 767
140
141     // if statement breaks down the "color" into three ranges:
142     if (color <= 255) // RANGE 1 (0 - 255) - red to green
143     {
144         redIntensity = 255 - color; // red goes from on to off
145         greenIntensity = color; // green goes from off to on
146         blueIntensity = 0; // blue is always off
147     }
148     else if (color <= 511) // RANGE 2 (256 - 511) - green to blue
149     {
150         redIntensity = 0; // red is always off
151         greenIntensity = 511 - color; // green on to off
152         blueIntensity = color - 256; // blue off to on
153     }
154     else // RANGE 3 (>= 512)- blue to red
155     {
156         redIntensity = color - 512; // red off to on
157         greenIntensity = 0; // green is always off
158         blueIntensity = 767 - color; // blue on to off
159     }
160
161     // "send" intensity values to the Red, Green, Blue Pins using analogWrite()
162     analogWrite(RED_PIN, redIntensity);
163     analogWrite(GREEN_PIN, greenIntensity);
164     analogWrite(BLUE_PIN, blueIntensity);
165 }

```

REQUIRED CHALLENGES:

1. Add your own function to the program that will just display a solid olive color LED.
2. Add your own function to the program that will cycle through 8 different non-standard colors. Make sure you indicate in your comments what the colors are. You may need to refer to a RGB color chart.
3. Be Creative! Think of something else you can program with the RGB LED and "just do it!"