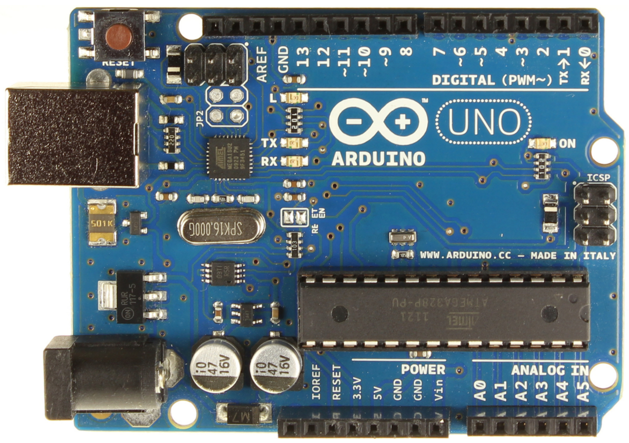
FabShops

Introduction to Arduino Prototyping

By: Adrian Yao, Thor Walker, & Sal Testa



What is a Microcontroller?

A **microcontroller** is a small computer built on an integrated circuit (IC) chip designed to be used in embedded systems and integrated devices. In contrast to a **microprocessor** in personal computers, microcontrollers are used to automatically control products and devices. Examples include the control system for car engines, remote controlled devices, 3D printers, power tools, toys, etc. As the cost to manufacture electronics continues to decrease, microcontrollers are becoming ubiquitous in our daily lives, and it is important to be able to prototype with them.

What is inside of a microcontroller?

There are many different microcontroller packages in the market designed for various applications, but all microcontrollers are similar in that they have a **central processing unit (CPU)**, **program memory**, **data memory**, and programmable **input and output (I/O)** peripherals. While some microcontrollers have internal clocks, most setups require an external clock component separate from the IC package, as they tend to be more reliable. Below is a block diagram of the basic components of a microcontroller.

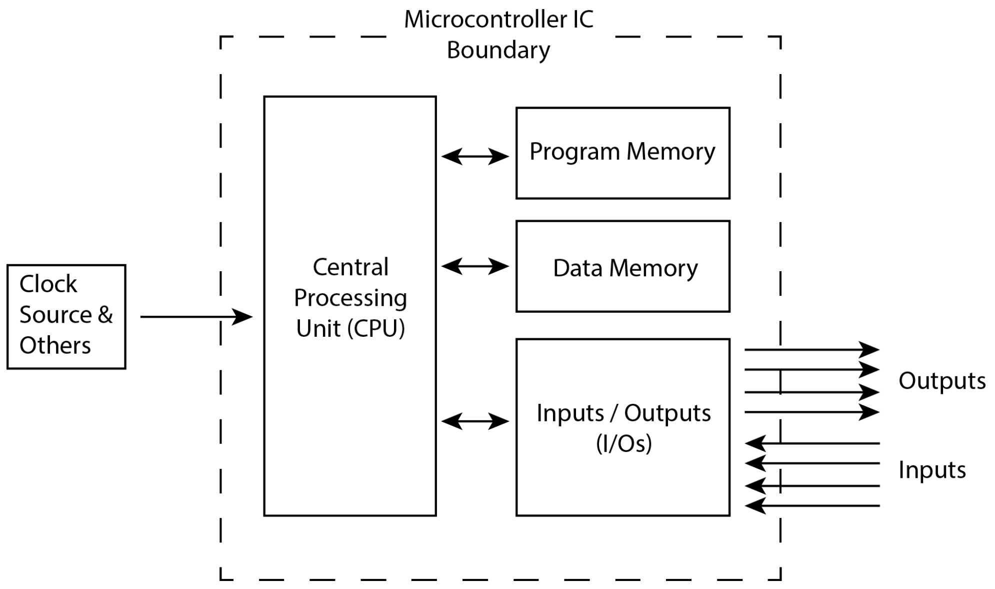


Figure 1: The basic internal and external components to a microcontroller system

Central Processing Unit (CPU)	This is the brain of the microcontroller. All computations occur within this unit in the IC package in the form of 1s and 0s.
Program Memory	A microcontroller will have two types of memory: **non-volatile memory** and **volatile memory**. In this case, the word "volatile" refers to the permanence of the memory, and differs in that volatile memory is not preserved when the power source is disconnected. **Program memory is non-volatile memory**, and it is the set of instructions for the CPU to carry out. It is not erased when the power source is disconnected, and thus allows machines and devices to be turned on/off while still "knowing" what to do. They are, however, able to be erased electrically if a new set of commands is to be uploaded to the microcontroller. These may often be referred to as **Read-Only Memory (ROM)** or **Electrically Erasable Programmable Read-Only Memory (EEPROM)**.
Data Memory	**Data memory is volatile memory**, and it usually consists of the saved values of short-term variables or information from inputs and output. They are not preserved when the microcontroller is "reset" or disconnected from power, and are referred to as **Random Access Memory (RAM)**.

Inputs & Outputs (I/Os)	The inputs and outputs are what make the microcontroller useful as it can interface with its surrounding environment using sensors and actuators. I/Os can be in two types: **digital** and **analog**. Digital I/Os consist of information given in 1s and 0s while analog I/Os consist of measurable electrical signals often translated into a range of values as a byte. Examples: Digital Input: On/Off pushbutton Digital Output: LED on/off Analog Input: Temperature sensor feeding a range of values Analog Output: Feeding LED varying voltages between 0 and 255 to fade in/out
External Clock Source	The CPU of the microcontroller works off of sequential logic that inherently requires a clock source to synchronize its operations. Thus, an external clock source is needed to provide a reliable reference of time. These are often quartz crystal oscillators or ceramic resonators.

What is an Arduino?

An Arduino board is simply a platform where a microcontroller is already connected with input and output pins, an external clock source, and where program memory is easily modified using a USB connection with a computer. At the heart of it, it is nothing more than the block diagram seen in *Figure 1*.

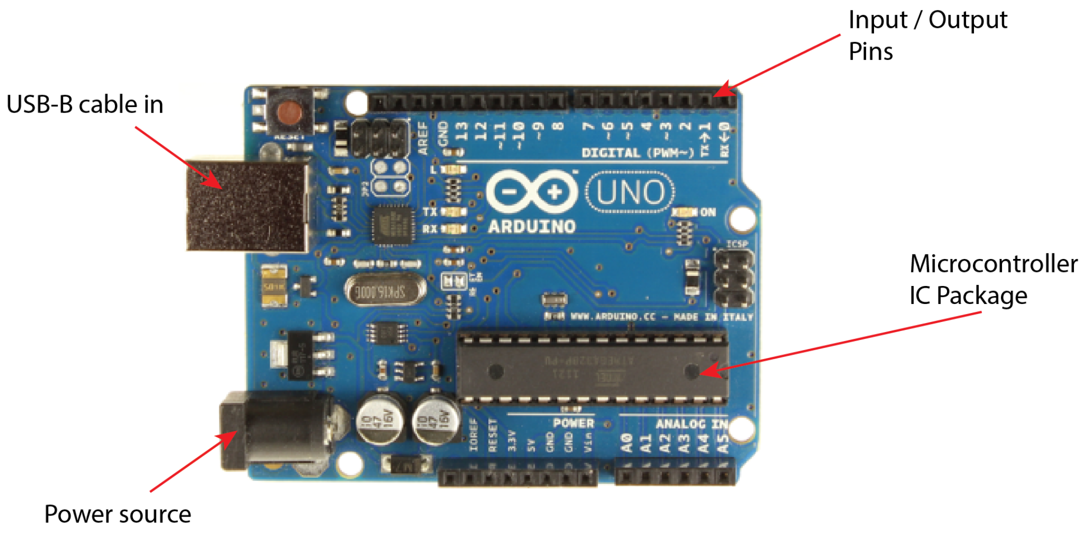


Figure 2: Key components on an Arduino Uno board

What is the Arduino IDE?

The Arduino Integrated Development Environment (IDE) is simply a free software developed by Arduino to program the microcontroller. While it is a unique programming language (that is very easy to learn), it is merely a translator that translates commands into 1s and 0s for the microcontroller to execute. A microcontroller can be programmed using various languages, but the end result will all be 1s and 0s that do the same thing. In fact, you can use the Arduino platform to program individual microcontroller IC packages that can be used in dedicated printed circuit boards elsewhere completely independent from Arduino. Thus, you can view the Arduino IDE as simply a middleman between you and the microcontroller.

Why Arduino?

You may be wondering what all the hype is about Arduinos. Arduinos are not unique, as there are other readily-interfaceable microcontroller boards out there. However, the combination of the high-quality Arduino boards, the fact that they are open source, the simple and free IDE, the ease of uploading commands, and the strong Arduino online community make it a standard platform for hobbyists, makers, and engineers to prototype with.

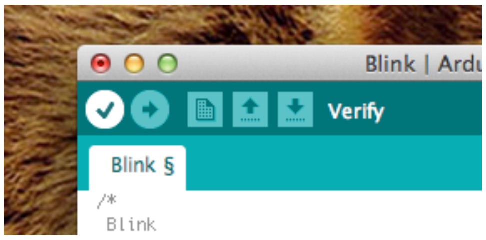
An Introduction to the Arduino IDE

Basics

The code works by running the setup method once, and then calling the loop function over and over forever (or until the microcontroller loses power).

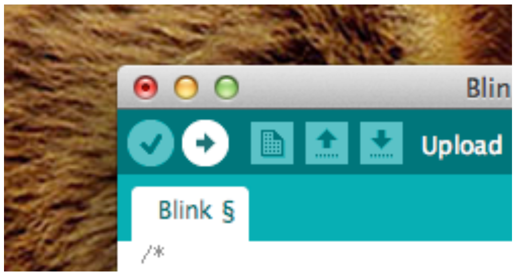
```
void setup()
{
  // put your setup code here, to run once:
}

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



```
/*
 Blink
```

Verify makes sure that your code is correct and will run without uploading it.



```
/*
```

Upload loads the code onto the Arduino.

Pin Modes

The pins on the Arduino can either receive input or output a signal.

```
int RED_LED = 13;

void setup()
{
  pinMode(RED_LED, OUTPUT);   // set the 13 pin to output
  digitalWrite(RED_LED, HIGH); // set the 13 pin high
}

void loop()
{
}
```

The example code above sets a pin to output and then sets it high thereby powering an LED.

```
int LED = 10;
int BUTTON = 3;

void setup() {
	pinMode(LED, OUTPUT);
	pinMode(BUTTON, INPUT); // input mode
}

void loop() {
	int readVal = digitalRead(BUTTON); // read the button
	if(readVal == HIGH){
		digitalWrite(LED, HIGH); // on if the button is down
	} else {
		digitalWrite(LED, LOW); // off otherwise
	}
}
```

By setting the pin to input, we can see whether the button is down and then turn on the light accordingly.

Arduino Projects for your Learning

Now that you have a better understanding of how the Arduino sends and receives information, you can now go ahead and begin making some cool projects using the Arduino!

This packet developed by the FabShop Coordinators is designed for you to learn the basics of Arduino programming through a series of projects that are about 75% complete. With detailed comments and descriptions provided explaining the reasoning behind the code, you should be able to continue and finish off the last few lines of code and get the projects working!

There are eight projects included in this packet complete with descriptions and wiring diagrams, and they vary in difficulty. If this is your first time working with Arduinos, we recommend you start off with the first project and work your way through the packet at your own pace. However, if you would like to take up the challenge, feel free to jump to any project that you'd like to try out.

The eight projects include:

1. Blinking an LED on and off	Simple output
2. Controllable RGB LED	More complicated output
3. Turning an LED on and off using a simple push button	Simple input and output
4. An automatic night light with adjustable brightness	Analog input and output
5. A sensor that assists garage parking	More complicated input and output
6. A 2-axis laser pointer controlled by the computer keyboard	One-way serial communication
7. A Simon Says LED game played by using the computer	Two-way serial communication

Getting Started on the Arduino IDE

1. Download the Arduino IDE software from the link: http://arduino.cc/en/Main/Software
    a. The latest version is 1.0.5, and there are links to download the software on both Mac and PC

2. Open the Arduino IDE

3. **To establish connection** between Arduino and Computer using USB-B cable, plug the USB-B cable to the Arduino and connect it with any one of the USB ports on your computer. Then go to **Tools > Serial Port** and choose either **/dev/tty.usbmodem1421** or **/dev/cu.usbmodem1421**

4. **To ensure that you are setup to program the correct board**, go to **Tools > Board** and select the board that you are using. The Arduino boards provided by FabShops is the standard **Arduino Uno**. Make sure there is a check mark next to the board you are using

5. **To check programs for errors** before uploading, click the top left button with a check mark ✓

6. **To upload programs to the board**, click the button with an arrow on the top bar →

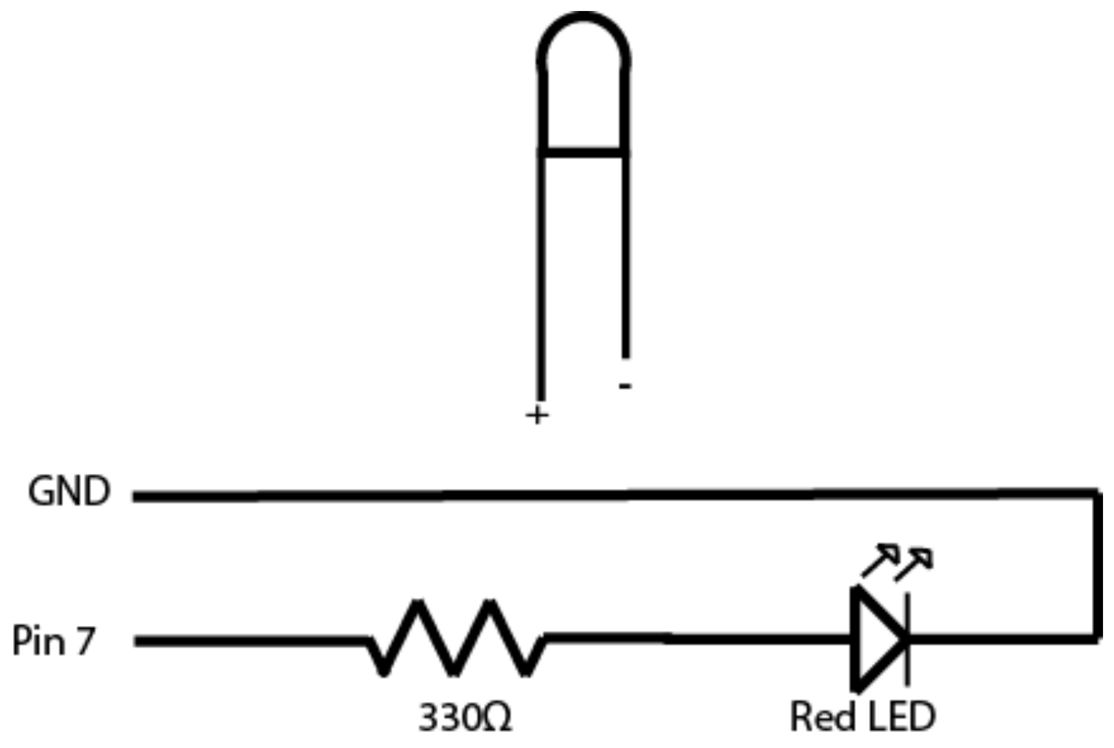
Blinking an LED On/Off

Description:
This is the most basic Arduino program, and all it does is to get an Arduino to blink an LED at a certain frequency.

What you will learn:
- Basics of digital outputs on the Arduino
- Basics of loops

What you will need:
- One 5mm LED
- One 330Ω resistor

Schematic:



While this project says that you need an LED and a resistor, the Arduino actually has an on-board surface-mounted LED that you can use that is already hooked up to an internal resistor. Thus you could replace pin 7 above with pin 13, as that is the pin associated with the on-bard LED, and not even worry about the resistor.

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

int led = 7;                          // this sets a variable of “integer” type labeled “led” to be a value of 7. This number will later be
                                      used to call on pin number 7 on the Arduino


void setup()                          // the void setup routine runs once when you press reset. It is a preliminary setup protocol
                                      that does not loop
{
  pinMode(led, OUTPUT);               // initialize the digital pin as an output
}

void loop()                           // the void loop routine runs over and over again forever
{
  digitalWrite(led, HIGH);            // turn the LED on (HIGH is the voltage level)
  delay(1000);                         // wait for a second. The delay function is calculated in milliseconds, thus 1000 means 1 sec.
  digitalWrite(led, LOW);             // turn the LED off by making the voltage LOW
  delay(1000);                        // wait for a second
}
```

Controllable RGB LED

Description:

A single tri-color LED is connected to the Arduino and wired on the breadboard. The Arduino will send it a series of patterns so that different colors fade on, fade off, and blink. A portion of the code is provided below.

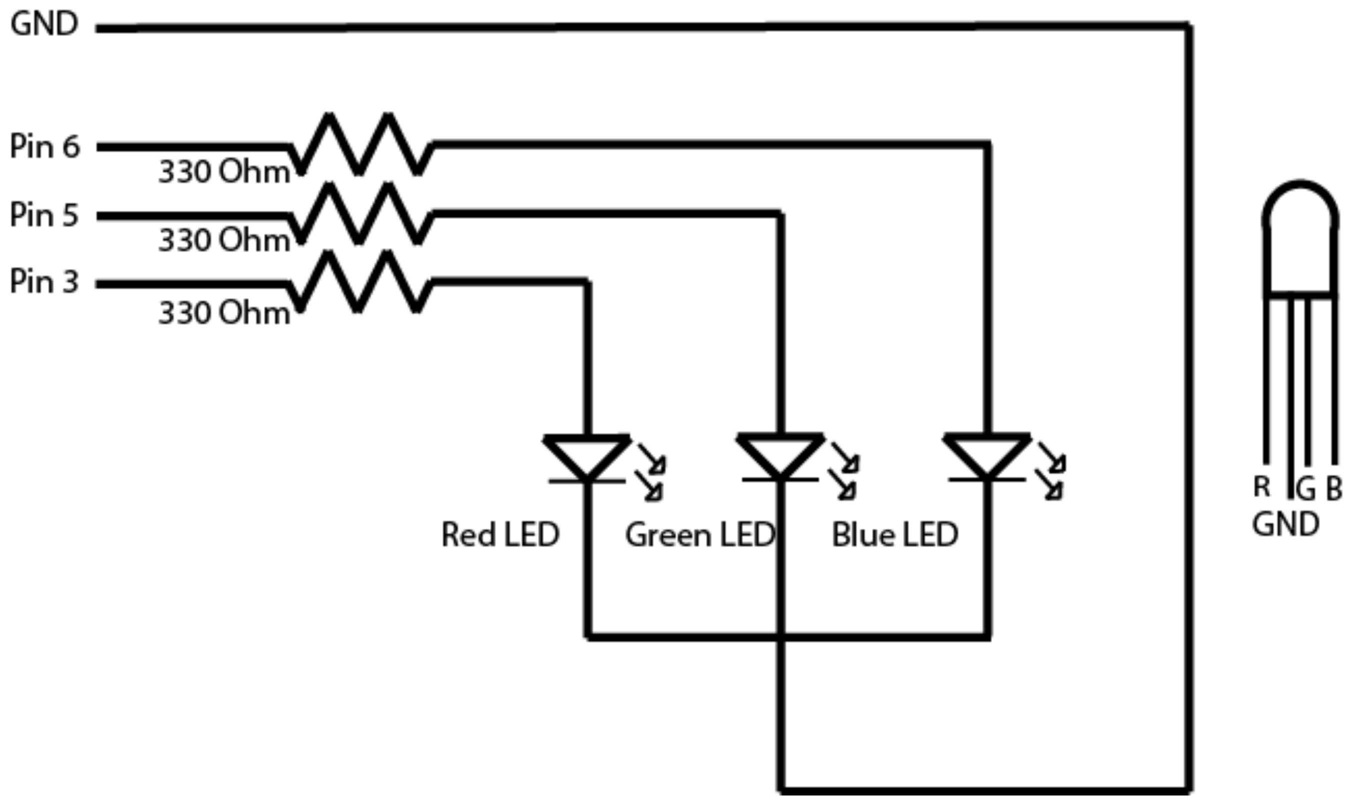
What you will learn:

- Sending analog outputs
- For loops and delays

What you will need:

- One tri-color LED
- Three 330 Ohm resistors

Schematic:



```
/* RGB LED Strip Patterns
 *Loops through different patterns on a tri-color LED with Red, Green, and Blue lights.
 *Pin 3 connects to 330 Ohm resistor in series with Red cathode
 *Pin 5 connects to 330 Ohm resistor in series with Green cathode
 *Pin 6 connects to 330 Ohm resistor in series with Blue cathode
* The Common (GND) cathode on the LED should be connected to Ground
 */


int R = 3;       // assign the integer value 3 to the variable R
int G = //?
int B = //?

int FADE = 12;    // milliseconds of delay within each pattern

void setup() {
  // set all the LEDs off initially
  analogWrite(R,//??);// send 0 voltage to pin 3 (red led)
                    // send 0 voltage to pin 5 (green led)
                    // send 0 voltage to pin 6 (blue led)
}

void loop() {
  int i;                    // create a counter 'i'
  int ON = 255;              // set the value of ON to maximum (255)
  int OFF = 0;               // set the value of OFF to zero

  // fade on the Green and Red LED'S simultaneously
  for (i = 0; i< 256; i++) { // incrementally increase i from 0 to 255
    analogWrite(R,i);        // send value of i to pin R
    analogWrite(G,i);        // send value of i to pin G
    delay(FADE);             // wait the value of FADE until next iteration of loop
  }

  // fade off the Green and Red LED's simultaneously
  //???
  }

  // blink the Green, Red, and Blue LED's in series
  int cnt;                           // set up a counter for the following loop
  for (cnt = 0; cnt < 50; cnt++) {    // run 50 loops of this cycle

    // turn red on, green off, wait 250 ms, turn red off, green on, wait 250 ms, green off
    analogWrite(R,ON);
    analogWrite(G,OFF);
    delay(250);
    //???
  }


//run 50 loops of a cycle
//????

  // Each loop of the cycle, turn blue on, wait 25 ms, turn blue off, wait 25ms

}
```

Turning an LED On/Off using a simple push button

Description:

A single pole, single throw (SPST) button is used to turn the pin 13 LED on the Arduino on or off. Pressing the button once will toggle the LED on or off. It does this by remembering the last state of the button circuit, and turning the LED on if the state switches. A portion of the code is provided below.

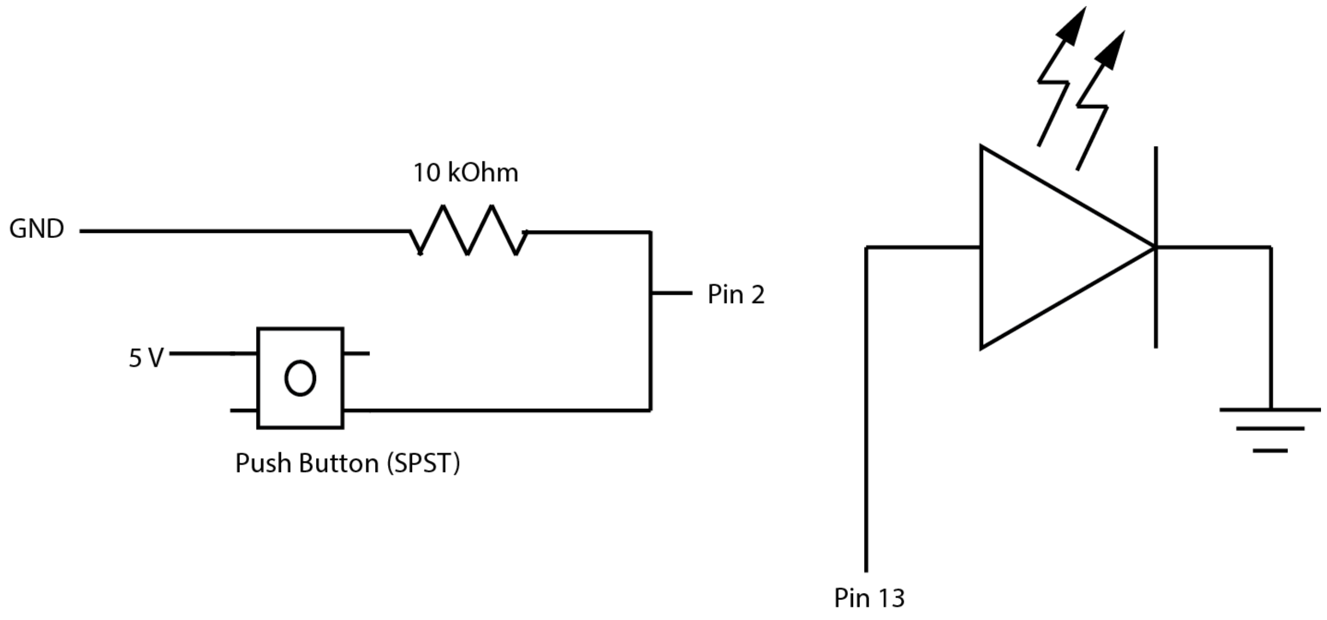
What you will learn:

- Reading digital input
- Outputting digital response
- Storing variables to make decisions in loops

What you will need:

- 1 SPST button
- jumper wires
- One 10k resistor

Schematic:



```
/*
 Button debounce
 Each time the input pin goes from LOW to HIGH (e.g. because of a push-button
 press), the output pin is toggled from LOW to HIGH or HIGH to LOW.  There's
 a minimum delay between toggles to debounce the circuit (i.e. to ignore
 noise). This is used to turn on and off a light emitting diode(LED) connected to digital
 pin 13

 The circuit:
  * LED attached from pin 13 to ground (already integrated into Arduino Uno
  * pushbutton attached to pin 2 from +5V
  *  10K resistor attached to pin 2 from ground

 */

// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;                    // the number of the pushbutton pin
const int ledPin = 13;                      // the number of the LED pin

// Variables will change:
int ledState = HIGH;                        // the current state of the output pin
int buttonState;                            // the current reading from the input pin
int lastButtonState = LOW;                  // the previous reading from the input pin

// the following variables are long's because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long lastDebounceTime = 0;                  // the last time the output pin was toggled
long debounceDelay = 50;                    // the debounce time; increase this if the output flickers

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);

  // set initial LED state
  digitalWrite(ledPin, ledState);
}

void loop() {
  // read the state of the switch into a local variable:
  int reading = //?????

  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH),  and you've waited
  // long enough since the last press to ignore any noise:

  // If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for longer
    // than the debounce delay, so take it as the actual current state:

    // if the button state has changed:
    if (reading != buttonState) {
      buttonState = reading;

      // only toggle the LED if the new button state is HIGH
      if (///?????) {
        ledState = !ledState;
      }
    }
  }

  // set the LED:
  ///????

  // save the reading.  Next time through the loop,
  // it'll be the lastButtonState:
  lastButtonState = reading;
}
```

An automatic night light with adjustable brightness

Description:

The purpose of this lab is to build a night light that automatically turns on when the room gets dark, and then have its brightness be adjusted with a potentiometer (variable resistor). The idea is to learn how to read and write analog inputs and outputs.

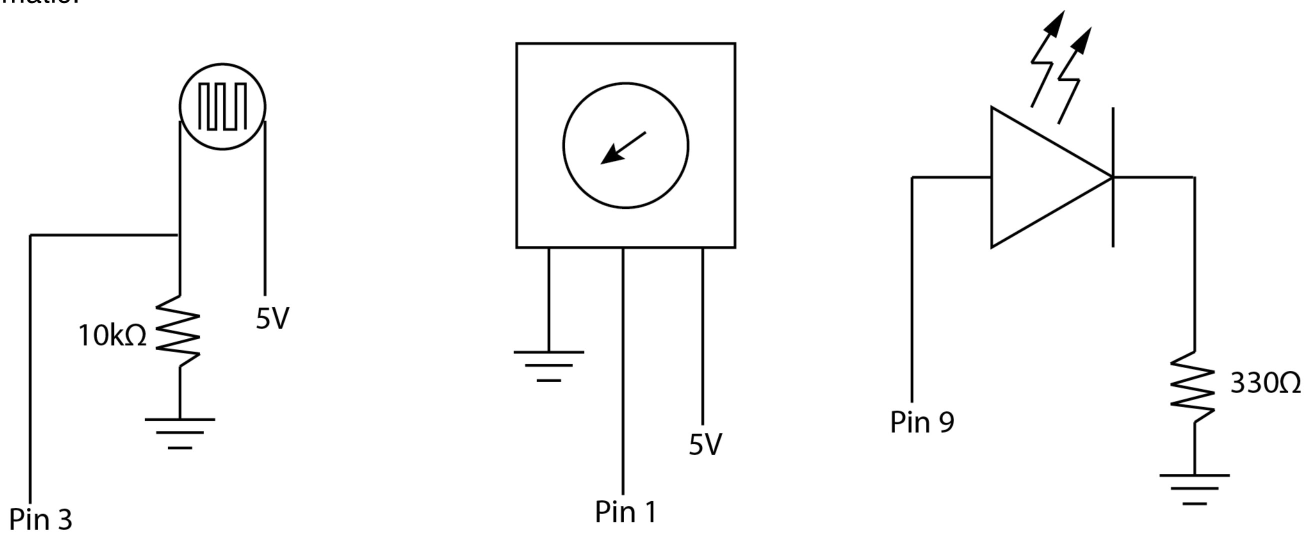
What you will learn:

- Working with analog signals
- How a potentiometer works
- How a photoresistor works

What you will need:

- One 10kΩ Trim potentiometer
- One 330Ω resistor
- One 10kΩ resistor
- One 5mm LED of your choice
- One photoresistor

Schematic:



The key behind this build is in the analog ins and outs that we will need to read and write. First, the photoresistor is merely a resistor that changes in resistance based on the ambient light. It's resistance decreases with light, and so we can take advantage of that to use it as a light sensor. By wiring it up in the configuration above, the Arduino will read the varying voltage signals and output an analog value between 0 and 1023. At a certain level of brightness, we can have the LED turn on.

However, we would also like to have an adjustable brightness LED, so we utilize a commonly used component called a potentiometer. A potentiometer is simply a variable resistor where you can physically change the resistance by turning a knob. If we use the same method to read the varying voltages through it using the Arduino, we can use those values to control the brightness of the LED.

Since LEDs are diodes and prefer to be either on or off, we connect the LED pin to pin 9 on the Arduino where you will notice that it has a ~ next to the number. This means that this pin is capable of pulse width modulation (PWM) where you pulse the LED on and off at varying frequencies to make it appear as if it were brighter/dimmer. This is still achieved, however, with the analogWrite() function.

```
const int potPin = 1;                     // const int sets constant integer variables that will not change. These are good for pin variables
const int ledPin = 9;
const int sensePin = 3;

int pot_val;                              // the value that will be read from the potentiometer
int sense_val;                            // the value that will be read from the photoresistor
int brightness;                           // the brightness value that will be used later on

void setup()
{
 pinMode(ledPin, OUTPUT);
 pinMode(potPin, INPUT);

 Serial.begin(9600);              // start serial communication with computer at 9600 baud rate (see below or ask for help from instructors)
}

void loop()
{
 sense_val = analogRead(sensePin);    // reads the analog output of the photoresistor
 delay(100);                              // delays the reading so you are not reading the values continuously
 Serial.println(sense_val);               // this sends the value read by the photoresistor to the computer and it is a value between 0 and 1023.
                                          // Open the Serial Monitor by going to Tools > Serial Monitor, and you will see the values read
                                          // pop up. Experiment with the photoresistor by covering up the photoresistor to light,
                                          // and figure out what value your threshold should be for when the LED should turn on

 if (sense_val <= VALUE??)                // use that value you found above to put in here
 {
  pot_val = analogRead(potPin);
  brightness = map(pot_val,0, 1023, 0, 255);          // this scales the values between 0 and 1023 to 0 and 255 since the Arduino can only
                                                      // handle 8 bits of information while most analog sensors output 10 bits of
                                                      // information
  analogWrite(ledPin, brightness);
 }

 else                                     // this turns the LED off in all other cases
 {
  analogWrite(ledPin, 0);
 }

}
```

A sensor that assists garage parking

Description:

The purpose of this project is to build a sensor that can assist you when you are parking your car in the garage. When space is cramped in the garage, there is often an optimal distance away from the wall that you should park your car, and sometimes it is hard to park the car exactly in that position. A device using an ultrasonic range sensor and a couple of LEDs could easily solve this problem by telling you exactly when you're close enough to the wall—eliminating the frustration of not parking close enough to the wall or the fear of bumping into the wall.

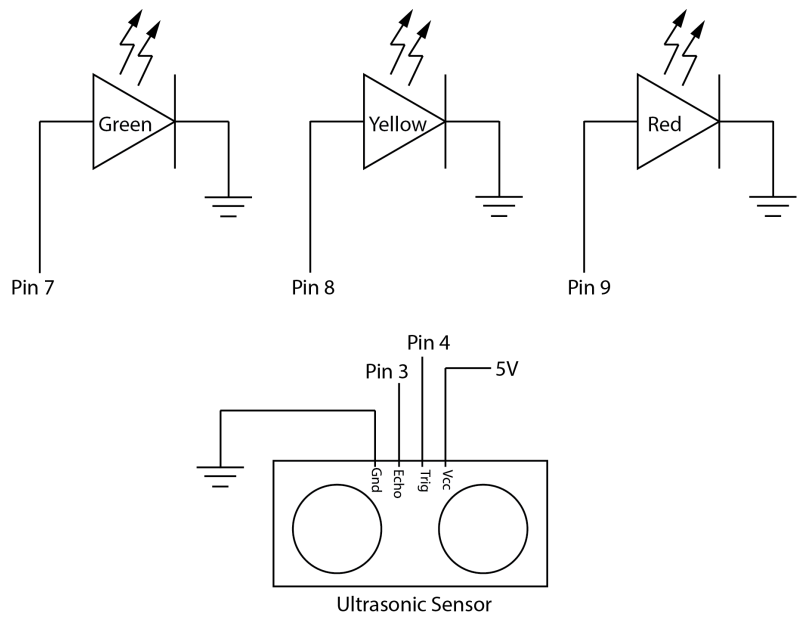
What you will learn:

- How an ultrasonic sensor works, and how to work with one
- How to use while loops

What you will need:

- 1 Ultrasonic range sensor
- 3 different 5mm LEDs (green, yellow, and red)

Schematic:



The key component of this device is the ultrasonic sensor. The way the ultrasonic sensor works is that it has a transmitter and a receiver. The transmitter (or trigger) will first emit an ultrasonic pulse. When the ultrasonic pulse bounces off of an object in front of it and reflects back to the sensor, the receiver (or echo) will calculate the amount of time it took for the ultrasonic pulse to travel to and from the object it was reflected off of. Dividing this time by a constant will give the distance in a unit of length. Given that sound waves travel at 340 m/s, figure out what this constant is. Then using this value, conditionals can be placed on the LEDs to communicate the distance that the car is away from the wall of the garage. In this device, we have the green LED light up when the car is within 2 meters from the wall but > 1 meter away. Then, when the car gets closer, the yellow LED will light up, until finally, the red LED will flash rapidly when the car is within 30 cm from the wall.

The following code does all of the above. However, since we don't want the red LED to be flashing forever when the car is finally parked, can you modify the code to make it so that the red LED flashes for only when the distance between the car and the wall is changing?

Hint: think about storing values to compare with new values. Look at the simple push button LED example.

```
const int echoPin = 3;
const int trigPin = 4;
const int greenLedPin = 7;
const int yellowLedPin = 8;
const int redLedPin = 9;
int maximumRange = 200;                         // sets the maximum range of the sensor to be at 200cm in which the green LED will light up
int yellowRange = 100;                        // sets the distance in which the yellow LED will light up
int redRange = 30;                           // sets the distance in which the red LED will light up
long duration;                              // creates a "long" variable that can store more bits than an "int" can. This stores the duration of time
                                            between the trigger and echo pulses sent by the ultrasonic sensor
long distance;
int soundConstant = ???;                     // figure out this sound constant based on the speed of sound




void setup()
{
  pinMode(greenLedPin, OUTPUT);
  pinMode(yellowLedPin, OUTPUT);
  pinMode(redLedPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(trigPin, OUTPUT);
}




void loop()
{
  digitalWrite(trigPin, LOW);               // the following five lines pulses the trigger pin or ultrasonic transmitter for a set amount of time
  delayMicroseconds(2);                     // described in microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);   // the pulseIn command calculates the time it takes for the receiver to receive a "HIGH" signal from the
  distance = duration/soundConstant;  // echo pin. The distance in centimeters is then calculated by dividing the time by a constant you found
                                            // above.

  if (distance > maximumRange)              // this turns all LEDs off in the case where the car is more than 2 meters away from the sensor so the
  {                                         // green LED is not always on when the car is not there
    digitalWrite(greenLedPin, LOW);
    digitalWrite(yellowLedPin, LOW);
    digitalWrite(redLedPin, LOW);
  }

  while ((distance <= maximumRange) && (distance > yellowRange))          // this turns the LED on when this condition is satisfied
  {                                                                       // note that the program does not exit the while loop until the
    digitalWrite(greenLedPin, HIGH);                                      // condition is no longer true
    digitalWrite(yellowLedPin, LOW);
    digitalWrite(redLedPin, LOW);

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = duration/soundConstant;
  }

  while ((distance <= yellowRange) && (distance > redRange))
  {
    digitalWrite(yellowLedPin, HIGH);
    digitalWrite(greenLedPin, LOW);
    digitalWrite(redLedPin, LOW);

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = duration/soundConstant;
  }
  while (distance <= redRange)
  {
    digitalWrite(redLedPin, HIGH);                                        // the following few lines flashes the red LED instead of just turning it on
    delay(100);
    digitalWrite(redLedPin, LOW);
    delay(100);
    digitalWrite(greenLedPin, LOW);
    digitalWrite(yellowLedPin, LOW);

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = duration/soundConstant;
  }
}
```

A 2-axis laser pointer controlled by the computer keyboards

Description:

The purpose of this project is to design a computer-controlled laser pointer that has two degrees of freedom using two servo motors. This is just a fun and cool project that you can show off to your friends. Since servo motors are an important part of mechanical prototyping, the point of this tutorial is to show you how to interface with these actuators. Another key point of this tutorial is showing you how you can use your computer as an input, and communicate with the Arduino through a serial port.

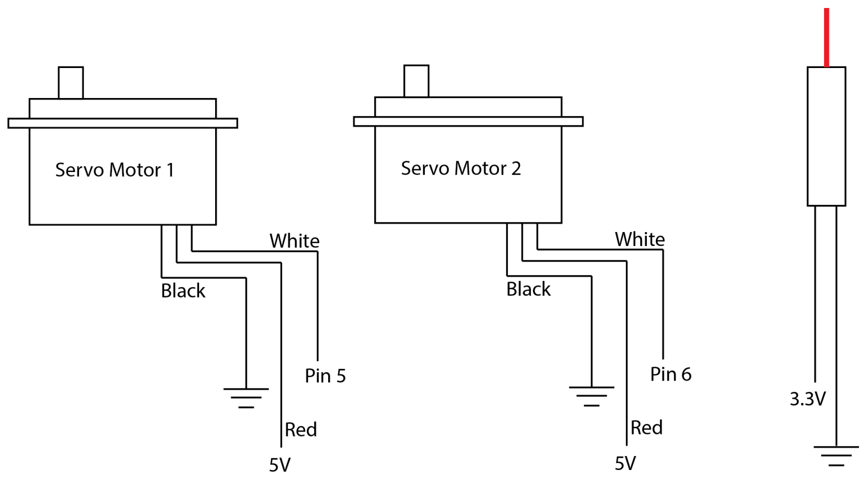
What you will learn:

- How to work with servo motors
- How to expand your Arduino base using pre-existing libraries
- How to communicate (one-way) with an Arduino using your computer through a serial port

What you will need:

- Two servo motors
- Hot glue
- Laser Diode

Schematic:



The beautiful thing about the Arduino platform is the pre-existing “libraries” that can be found open source on the internet. While some devices such as stepper motors or servo motors are difficult to program from scratch, you do not need to worry because the Arduino community realized that you do not need to reinvent the wheel if it is already done. Thus, there are a bunch of “libraries” that are pre-programmed functions into the Arduino. The Arduino default IDE already comes with a library to control servo motors, along with many others. If you happen to find yourself requiring a library for a new actuator/sensor in the future, just do a quick Google search and you should be able to find one.

The second cool thing about Arduinos is that you can communicate with them via the computer serial port, and give commands using your keyboard. Some Arduino boards (like the Leonardo and the Due) have more functionalities with the keyboard and mouse, whereas most Arduino Uno boards can only take single commands one at a time separated by an “enter” stroke in the Serial Monitor of the IDE.

The code provided below controls a laser diode (that works off of 3.3V, not 5V) using a two-axis servo motor control system. When you turn on the Serial Monitor in the IDE by going to Tools > Serial Monitor, you can then type in the “w,” “s,” “a,” and “d” commands followed by an enter stroke to send the laser pitching or yawing at 5 degree increments.

Can you program a way so that the laser homes back to center with the press of the “h” button?

```
#include <Servo.h>                                // this includes the Servo library already in the Arduino IDE

Servo yawServo;                                   // this lets the Arduino know how many servos you have and names them
Servo pitchServo;

int val = 90;                                     // the Servo library works with servos using degrees. The servos that you have
int pitch_old_val = 90;                           // have 180 deg. of motion, so this sets the initial pitch and yaw centered to 90 deg.
                                                  // it is labeled as "old value" because you will need to add/subtract degrees from the
                                                  // current state
char serialVal;                                   // this reads the characters "w" "s" "a" and "d" from the Serial Monitor
int yaw_old_val = 90;

void setup()
{
 yawServo.attach(5);                              // this is the same as pinMode for servos in the Servo library and designates the pin
 pitchServo.attach(6);

 Serial.begin(9600);                              // this begins communication with the computer through the serial port with a 9600
                                                  Baud rate.
}

void loop()
{
 while (Serial.available() == 0);                 // this tells the Arduino to do nothing while no input from the computer serial port is
                                                  detected

 serialVal = Serial.read();                       // reads the input from the computer
 Serial.println(serialVal);                       // echos what you just typed in back to you. You may notice that this actually outputs
                                                  a number instead of the letters that you're typing in. Can you explain this? Google
                                                  ASCII codes.


 if ((serialVal == 'w') && (pitch_old_val < 180)) // the following codes tell the servo motors to move in certain directions at 5 degrees
 {                                                // at a time when command keys are pressed while saving the new position as old_val
  pitchServo.write(pitch_old_val + 5);
  pitch_old_val = pitch_old_val + 5;
 }

 if ((serialVal == 's') && (pitch_old_val > 0))
 {
  pitchServo.write(pitch_old_val - 5);
  pitch_old_val = pitch_old_val - 5;
 }

 if ((serialVal == 'a') && (yaw_old_val > 0))
 {
  yawServo.write(yaw_old_val - 5);
  yaw_old_val = yaw_old_val - 5;
 }

 if ((serialVal == 'd') && (yaw_old_val < 180))
 {
  yawServo.write(yaw_old_val + 5);
  yaw_old_val = yaw_old_val + 5;
 }

 else                                   // this tells the Arduino to send back an error when a key other than "w" "s" "a" or "d" is pressed
 {
  Serial.println("Error: command not recognized");
 }

}
```

A Simon Says LED game played by using the computer

Description:

Simon is a classic memory game in which the player is shown a pattern of colors and then repeats the pattern from memory. This lab will demonstrate a more sophisticated logic where the microcontroller is doing a fair amount of work.

Example game:

1. intro sequence (line 82-98)
2. board blinks red
3. player presses red
4. board blinks red, green
5. player presses red, green
6. board blinks red, green, green
7. player presses red, green, red
8. board shows red and then starts over

At the beginning of each game, the Arduino determines the entire pattern sequence by randomly picking one of the LEDs for each position (line 120-122). The game starts by revealing more and more of the sequence as the player progresses. Once a full pattern is revealed, the Arduino waits for a button to change from not pressed to pressed (line 47-64) and then updates the game state (which button should come next). If the button pressed is not the one the game was expecting, then the game starts over.

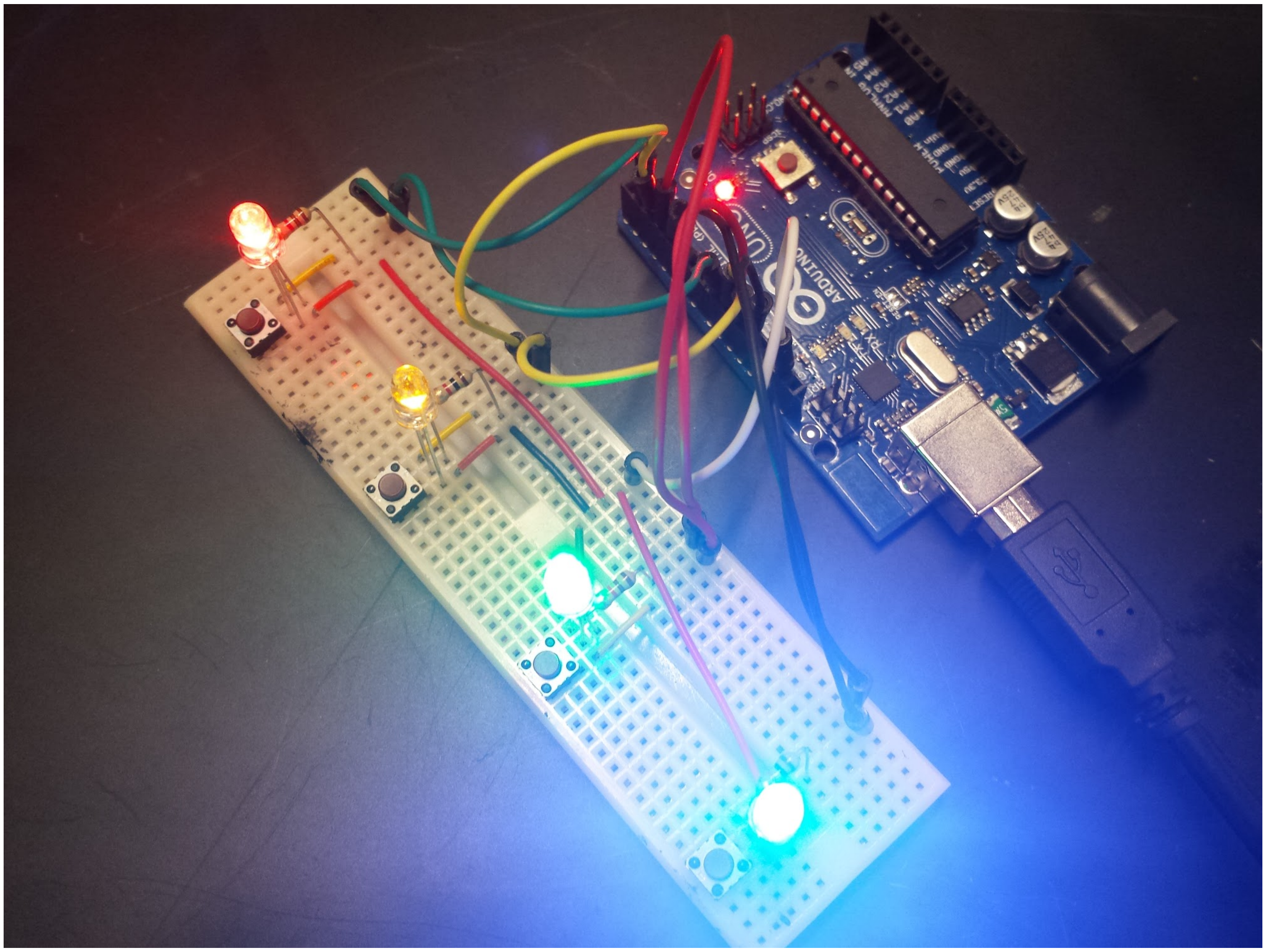
A few design decisions:

- The pattern length is set to 100. If someone has a really good memory, the code has undefined behavior for what would happen next. While the code could allow for a dynamic resizing of the pattern, there is no reason to be conservative with memory usage as this is the only process running on the chip.
- When a user presses a button, it could look like, to the Arduino, that the user pressed the button repeatedly thanks to a phenomenon called bouncing. This code gets around this by doing two things. First, it uses the pull-up (line 34) resistor which uses ground as opposed to power for the input signal. This decision was determined by testing against the default input type. Second, the Arduino waits one second after registering the button press (line 60) to give the bouncing a chance to die down and give the user a chance to remove their finger.

You will need

- Four LEDs
- Four 330 ohm resistors
- Four pushbuttons (borrow from your neighbor in necessary)

Schematic is not drawn but shown below in the photograph.



```
// pins corresponding to the LED anode
int leds[] = {8, 9, 10, 11};
// pins corresponding to the button
int buttons[] = {1, 2, 3, 4};
int NUM_BUTTONS = sizeof(leds)/sizeof(int);
// longest game possible
const int MAX_PATTERNS = 100;
int pattern[MAX_PATTERNS];
// last assigned pattern index
int finalIndex;
// pattern index being guessed
int currentIndex;
// a button's possible states
enum button {
        DOWN,
        UP
} button = DOWN;

/**
 * Initialize all the pins and play the opening sequence
 */
void setup() {
        randomSeed(analogRead(0));
        for(int i = 0; i < NUM_BUTTONS; i++) {
                pinMode(leds[i], OUTPUT);
                pinMode(buttons[i], INPUT_PULLUP);
        }
        start();
}

void start() {
        resetGame();
        intro();
        showPattern();
}

void loop() {
        for(int i = 0; i < NUM_BUTTONS; i++) {
                int readVal = digitalRead(buttons[i]);
                // Button not pressed
                if(readVal == HIGH){
                        off(i);
                        button = UP;
                // Button pressed
                } else if(readVal == LOW) {
                        on(i);
                        if(button == UP){
                                button = DOWN;
                                /**
                                 * Give the player time to remove their finger.
                                 */
                                delay(1000);
                                guess(i);
                }
            }
        }
}

void guess(int button) {
        // correct guess
        if(button == pattern[currentIndex]){
          currentIndex++;
            if(currentIndex > finalIndex) {
                currentIndex = 0;
                finalIndex++;
                showPattern();
            }
        // wrong guess
        } else {
            start(); // start over
        }
}

void intro() {
        // Cycle through all the LEDS
        for(int j=0; j<3; j++){
            for(int i = 0; i < NUM_BUTTONS; i++) {
                blink(i, 100);
            }
        }
        // Turn them all on
        for(int i = 0; i < NUM_BUTTONS; i++) {
            on(i);
        }
        delay(1000);
        for(int i = 0; i < NUM_BUTTONS; i++) {
            off(i);
        }

}

/**
 * Show the Simon sequence
 */
void showPattern() {
        for(int i = 0; i < NUM_BUTTONS; i++) {
            off(i);
        }
        delay(1000);
        for(int i=0; i <= finalIndex; i++){
          blink(pattern[i]);
          delay(1000);
        }
}

/**
```

```
 * Re-initialize all the values
 */
void resetGame() {
			finalIndex = 0;
			currentIndex = 0;
			for(int i=0; i<MAX_PATTERNS; i++){
			    pattern[i] = random(0, NUM_BUTTONS);
			}
}

void blink(int led) {
			blink(led, 1000);
}

void on(int led) {
			digitalWrite(leds[led], HIGH);
}

void off(int led) {
			digitalWrite(leds[led], LOW);
}

void blink(int led, int delaySpeed) {
			on(led);
			delay(delaySpeed);
			off(led);
}
```