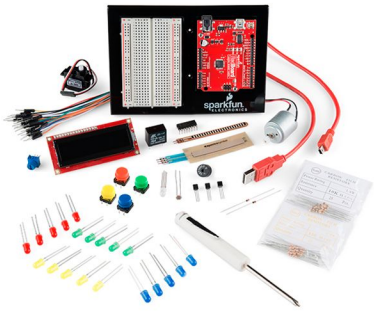
SparkFun Inventor's Kit Teacher's Guide to the Circuits

Draft -- July 2016



The SparkFun Inventor's Kit offers a great start to embedded electronics, programming, and engineering. Here we break down each of the circuit examples by concept / vocabulary to teach as well as a series of recommendations for using the SIK in your classroom.

As an effort to clean up the code, we have moved a large portion of the comments to a secondary "readme.h" file and ported these over to codebender, and on-line Arduino programming environment.

Revised circuit / code examples → sparkfun.com/sikcodebender

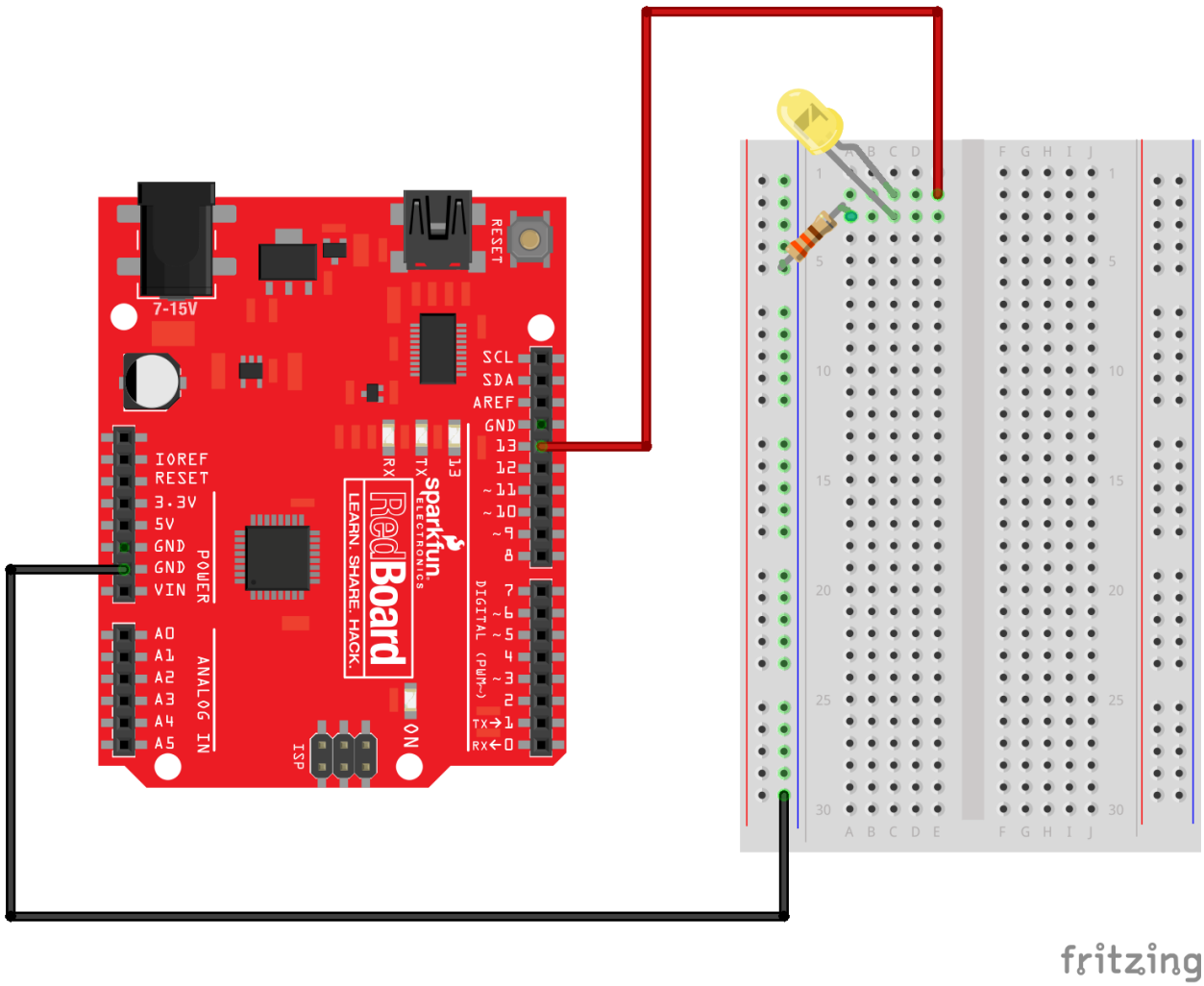
Table of Contents

(This page intentionally left blank.)

Circuit #1 - Blink

[example code](https://codebender.cc/sketch:77046) - https://codebender.cc/sketch:77046

This is the first project for physical computing. It is the equivalent of the "Hello World!" program that is often used to introduce people to programming in other languages. This project uses a single LED and a resistor and roughly 10 lines of code.



Learning objective(s):

1. basics of programming syntax and control.
2. Understand basics of breadboard usage.
   learn.sparkfun.com - tutorial: [How to use a Breadboard](#).
   giant breadboard poster -- [resource](#).
3. control of GPIO pins on an Arduino
4. commands / functions to introduce:
   a. `pinMode([pin], [INPUT/INPUT_PULLUP/OUTPUT]);`

The pinMode() command sets the mode for the general purpose I/O pins on the Arduino.

b. `digitalWrite([pin], [HIGH / LOW]);`
The digitalWrite() command sets the state of a pin. HIGH indicates that the pin will be ON and will output 5V. LOW indicates that the pin will be OFF and will output 0V.

c. `delay([time_milliseconds]);`
The Arduino runs with a 16 MHz clock. This means that it is 62.5 ns between clock cycles. To control the flow of the program, we can use the delay() command. The parameter in between parentheses is the delay in milliseconds.

Vocabulary / Concepts:

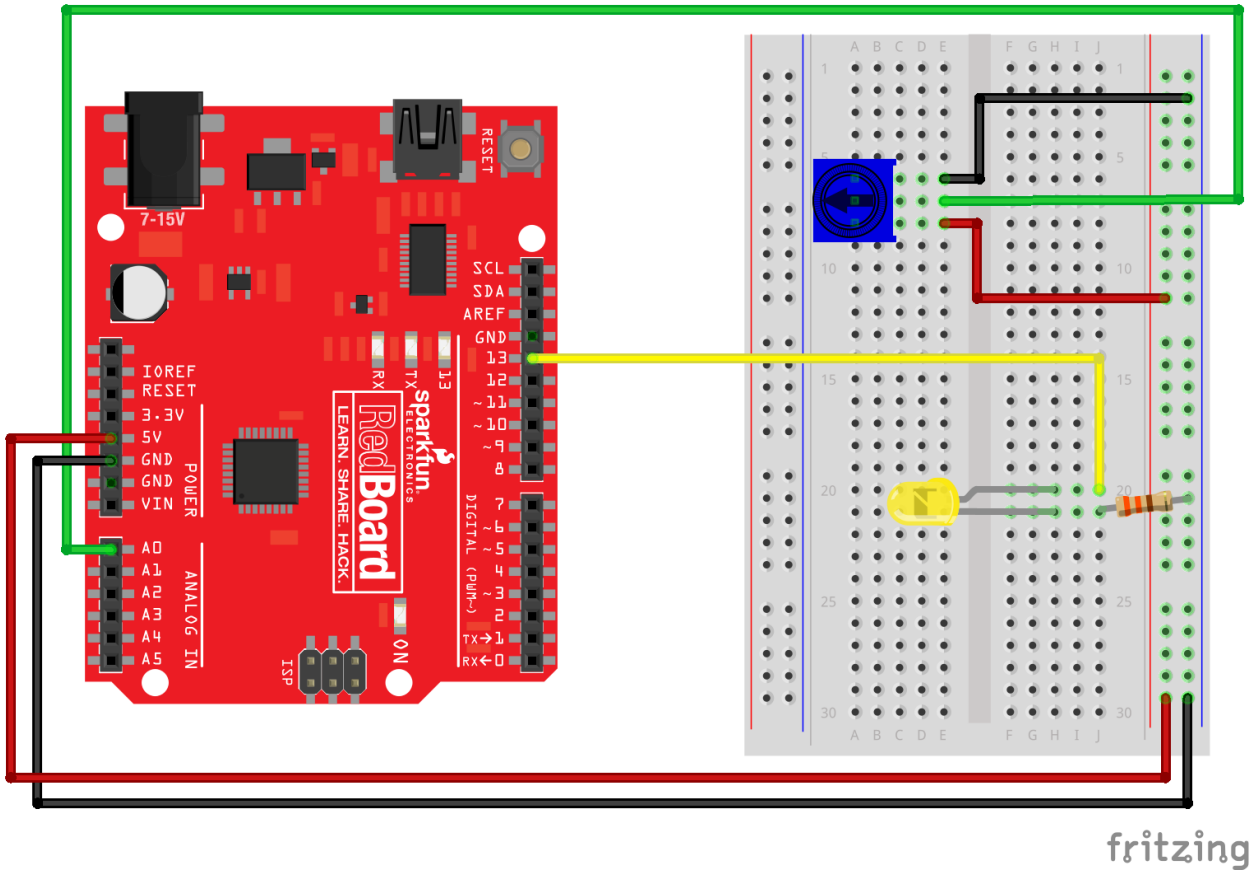
- **circuit** - A circuit is a complete loop which connects a power source, through a device, and back to the the power source.
- **LED** - Light emitting diode (LED) is a device which converts electrical energy into light energy. LEDs are polarized. They only work when they are connected in the correct direction. The long leg of a standard LED is usually the POSITIVE side. LEDs have very low resistance. LEDs have a maximum current of about 20 mA.
- **resistor** - A device which impedes or slows the flow of electricity. This is used in the circuit to limit current flow through the LED.
- **ground** (GND) - Ground is the return for current flow in a circuit. Ground refers to the negative terminal of the power supply on the Arduino.
- **upload** - Sending the program to the microcontroller.
- **compile** - converting the human-readable code into 1's and 0's that instruct the microcontroller how to behave and perform.
- **digital** - Digital refers to values that exist in only one of two states. Generally this is ON or OFF.
- **microcontroller** - Sometimes abbreviated µC, uC or MCU), a microcontroller is a small *digital* computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. It's the "brain" of the system.
- **pins** - Pins are the physical connections on the outside of the microcontroller. The "pins" are general purpose and can be either inputs or outputs to the microcontroller.
- **Arduino** - Arduino is the general term used to describe the microcontroller board and also programming language \ environment.
- **breadboard** - sometimes called a "solderless" breadboard, this is a prototyping tool that allows us to quickly connect wires together without soldering or twisting them together.

Key features of the breadboard to know are that the rows of 5 holes are all connected together. On the edges of the board are vertical power rails. These allow us to quickly connect multiple components to either 5V or GND. The power rail is one continuous vertical connection.

Circuit #2 - Potentiometer

[example code](https://codebender.cc/sketch:77047) - https://codebender.cc/sketch:77047

This example demonstrates using a trim potentiometer as a simple analog input to control the blinking rate of an LED.



Learning objective(s):

1. Apply and use variables in code.
2. Apply Ohms Law and a voltage divider circuit using a potentiometer.
3. Use a muti-meter to measure resistance.
   learn.sparkfun.com - tutorial: [How to use a Multimeter](#).
4. Understand analog to digital conversion / translation from voltage to data. What is the difference between analog and digital?
   learn.sparkfun.com - tutorial: [Analog vs. Digital](#)
5. **commands / functions to introduce:**
   a. `int varName;`
      This line declares a variable. Declaring variables follows this general structure:
      `<data type> <variableName>;`
      There are several data types used in Arduino. `int` defines the variable as an integer and means the value can be any integer value from -32,768 to 32,767.

Other common data types include: `byte`, `char`, `long`, and `float`. These each use a different amount of memory and can represent different size numbers.

Variables generally initializes with the value of 0, but you can also set the initial value of a variable by using the assignment operator "=" as in `int delayTime = 500;`.

b. `const int constantName;`
`const <data type> <constantName>;`
Similar to variables, the keyword `const` declares this as a constant rather than a variable. These can be initialized with a value, but this value can not be manipulated or changed. These also require less memory space than a regular variable. http://arduino.cc/en/Reference/Const

c. `analogRead([pin]);`
The `analogRead()` function will read the voltage on one of the analog input pins (A0 - A5).

Vocabulary / Concepts:

- **variables** - variables are placeholders for a value or number used in the program. Variables can be used to store and manipulate numbers within a the program.

- **int** - int designates a variable as an integer. An integer represents a 16-bit signed number that ranges from -32,768 to +32,767 (-2^15 to +2^15 - 1)

- **analog** - analog refers to values or things which exist across a range. It differs from digital in that an analog value can take. For Arduino, we are able to read in an analog value using an analog to digital converter. Remember that the microcontroller is a digital device.
https://learn.sparkfun.com/tutorials/analog-to-digital-conversion

- **potentiometer** - A potentiometer is a 3-pin device also known as a variable resistor. For this device, the resistance between the two outside pins is fixed at 10kΩ, but the resistance between the center pin and the outside pins changes relative to the amount the knob is turned. When 5V and GND are applied to the two outside pins, the center pin will have a voltage that is divided relative to the resistance to GND. In short, the center pin's voltage will vary between GND and 5V as you turn the knob.

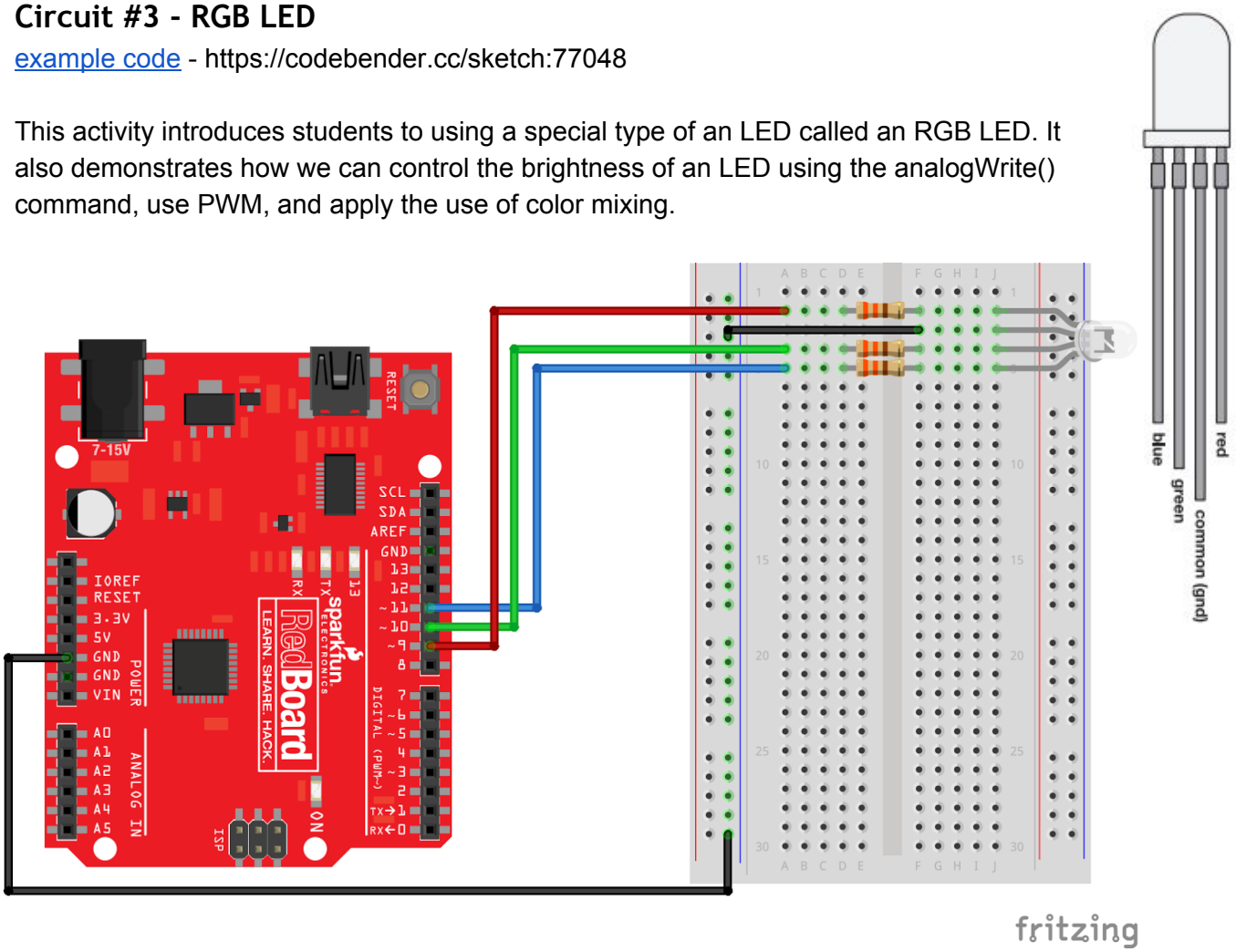
- **voltage** - Voltage represents the electrical potential energy in a system. It is analogous to the height of a water tower used to deliver water to a town.

- Ohms law: $V = I \bullet R \rightarrow$ voltage is **directly** proportional to the resistance.

Circuit #3 - RGB LED

[example code](#) - https://codebender.cc/sketch:77048

This activity introduces students to using a special type of an LED called an RGB LED. It also demonstrates how we can control the brightness of an LED using the analogWrite() command, use PWM, and apply the use of color mixing.



Learning objective(s):

1. Apply and use of "constants" in code.
2. Understand wiring / control of an Integrated LED circuit (RGB)
3. Understand digital to analog conversion using PWM
4. Understand calling functions.
5. Manipulating and writing custom functions used in Arduino.
6. **commands / functions to introduce:**
    a. const int constantName = 0;
    b. **analogWrite**([pin],value);

Vocabulary / Concepts:

- **analog OUTPUT** - Analog refers to something that can take on a range of values. For INPUTs, an analog INPUT is read using the analogRead() command. analog OUTPUTS on the Arduino vary across a range of values from 0 - 255. This corresponds to the average voltage of the pin by means of pulse-width-modulation.

- **PWM** - Because the microcontroller is purely a digital device, the only way it can provide an analog OUTPUT is by manipulating the duty cycle of a repeating square pulse. The frequency of the pulse is so fast (~490 to 980 Hz) that you can't see the LED flicker or blink. What is the time period for a PWM pin if the frequency is 490 Hz? How much delay is there between the ON and the OFF?
  https://learn.sparkfun.com/tutorials/pulse-width-modulation

- **common** - Common refers to the pin or connection that all things are connected to. Sometimes Ground is called the Common pin . Why is that?

- **cathode / anode** - The RGB LED is a common cathode LED. Looking at the diagrams in the guide, is the cathode positive (+) or negative (-)? [Cathode is negative and Anode is positive.]
  - There are some RGB LEDs that are common anode. What do you think that means? Can you draw a diagram for what this might look like?
- **color & color mixing** - with the RGB LED, we can create 256 shades of Red, 256 shades of Green, and 256 shades of Blue and every combination of the three. How many different colors is this? (256*256*256 = 16,777,216 colors!) Using an online tool like color selector like colorpicker.com to pick out different values of Red Greed and Blue.

- **functions** - functions are instructions or groups of instructions that are referenced by a name. In the beginning, we declare two functions for every sketch `setup()` and `loop()`. These functions each have a group of instructions that are captured by curly braces { } in the code.

  To simplify our code, we can also write our own custom functions. If the function does not return or output a value, it is declared using the keyword `void`. - like with `void setup()` and `void loop()`.

  In the main loop(), we see a line that says `mainColors();` This is referred to as a "function call" -- it calls the function `mainColors()` which is defined lower in the code. Each time the loop runs, it calls `mainColors()`. Notice that the other function call `showSpectrum()` is commented out. Remove the two // to un-comment out the showSpectrum();

```
void loop()
```

```
{
  mainColors();        // Red, Green, Blue, Yellow, Cyan, Purple, White
  //  showSpectrum();    // Gradual fade from Red to Green to Blue to Red
}

void mainColors()
{
 . . .
}
```

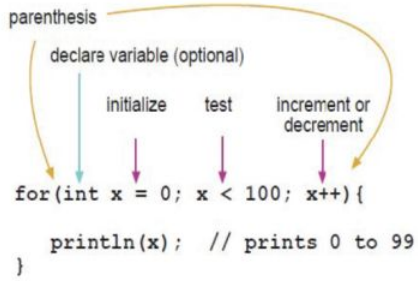
- **for()** loop - The for loop is a way to repeat a block of code a specific number of times using an index counter. Example:

  ```
  for(int index = 0; index < 10; index = index + 1)
  {
   . . .
  }
  ```

  The for() loops has three basic parts that are inside the parentheses. The first part is the declaration / initialization, the second is the condition (also called test), and the third is the increment / decrement.

  ```
  for(initialize; condition; increment)
  {
   . . .
  }
  ```

  The code inside the `for()` loop will repeat 10 times. The variable `index` will be initialized at 0, and index will be incremented by 1 each time it runs through the loop until it reaches 10. This will fail the condition and this will exit the loop and continue onto the next line of code following the `for()` loop.

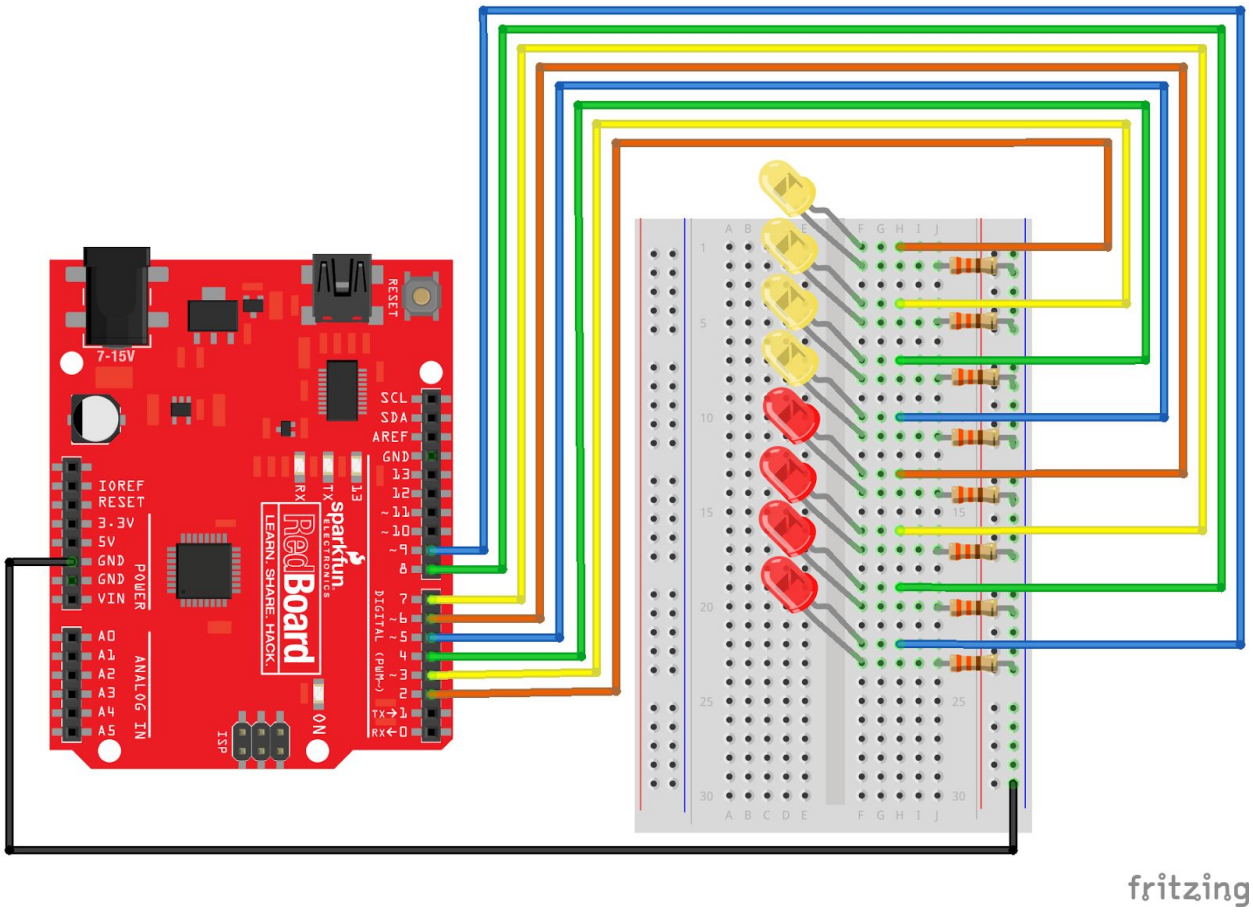


an example of the for() loop.

Circuit #4 - Multiple LEDs

example code - https://codebender.cc/sketch:77049

This example will show us how to wire up 8 LEDs and use a data type called an array to make controlling these LEDs easier. This circuit uses two different colored LEDs, but if you have extra LEDs, you can use 8 LEDs of the same color to create a better "bar graph" style effect.



Learning objective(s):

1. Applying and using an array variable.
2. Understand digital to analog conversion using PWM
3. Understand calling functions.
4. Understand syntax and usage of for loops
5. Manipulating and writing custom functions used in Arduino.
6. **commands / functions to introduce:**
    a. arrays -- int arrayName[] = {4, 3, 2, 1, 0};
    b. for()

Vocabulary / Concepts:

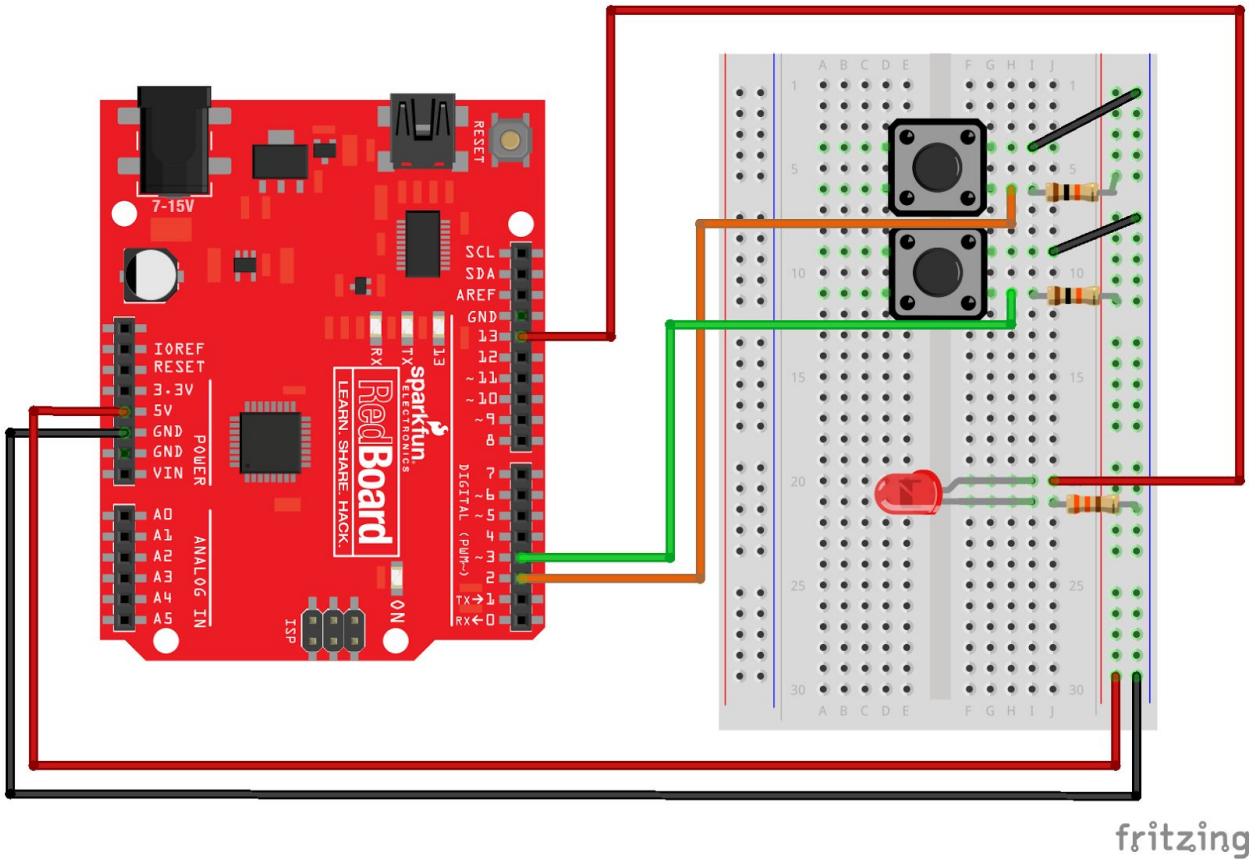
- **arrays** - An array is basically a list variable. Each of the items in an array are indexed by a number that is in brackets [ ] - following the name of the array. The first item in the list is always index 0. An example: arrayName[0] from above is equal to 4. arrayName[1] is equal to 3.
- **index** - An index is a number that is used to identify the item in an array. In Arduino, all arrays start at index 0 (meaning the first item in the list is referenced by [0].)
- **for loops** - For loops are perfect for working with arrays. Since we know how long an array is, we can easily index through an array with a simple for loop;

Circuit #5 - Push Buttons (original)

Sometimes we want to be able to start a circuit or select a state using a push button. This sketch demonstrates how we do this.

The original circuit #5 uses two buttons and is an exercise around compound boolean logic. When you push one button, the LED turns on. When you push the other LED, the LED also turns on. But, when you push both buttons, the LED does ***not*** turn on. This is the project is the one that is outlined in the printed SIK guide. I've made adjustments to this project using a single push button and an RGB LED as described in v2 below.

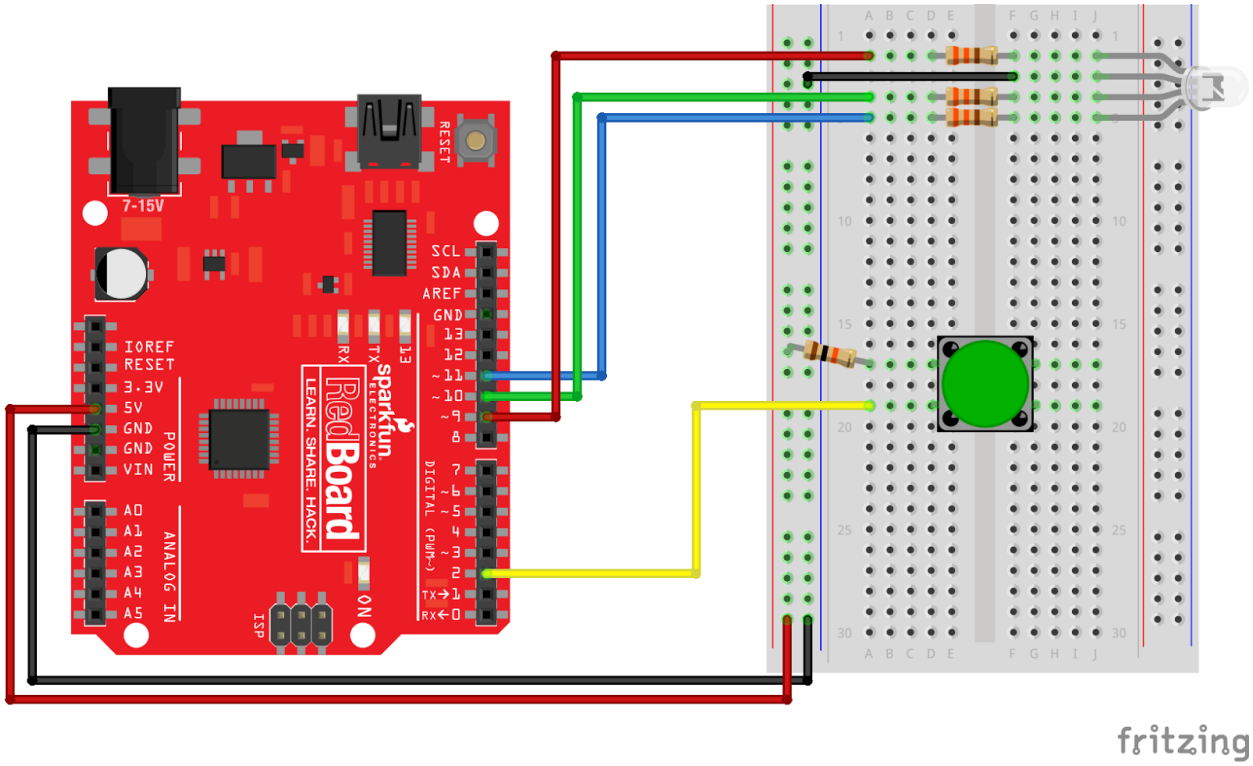
The original example code for this project can be found here:
[example code](https://codebender.cc/sketch:77050) - https://codebender.cc/sketch:77050



Circuit #5 - Push Buttons v2

Here is an alternate sketch / circuit to the one in the SIK. I'm calling it Circuit 5_v2. This one uses a single push button and an RGB LED. When you push the button the Arduino changes "modes" or what we call "states" in embedded electronics.

Circuit 5_v2 sketch - https://codebender.cc/sketch:128600



Learning objective(s):

1. Use boolean logic with an if() statement
2. Use of if() else()
3. Compound logic statements
4. Manipulating and writing custom functions used in Arduino.
5. Understand how to use a state machine to sequence through steps.
6. **commands / functions to introduce:**
    a. digitalRead();
    b. if(), if() else, if() else if() else
    c. boolean comparison operators: ==, >, <, >=, <=
    d. compound boolean operators: &&, ||, !

Vocabulary / Concepts:

- **boolean expression** - Boolean expressions are combinations of logic comparisons that is either TRUE or FALSE. Like mathematical operators, we can combine multiple operations together using compound operators: && (and), || (or), and ! (not).

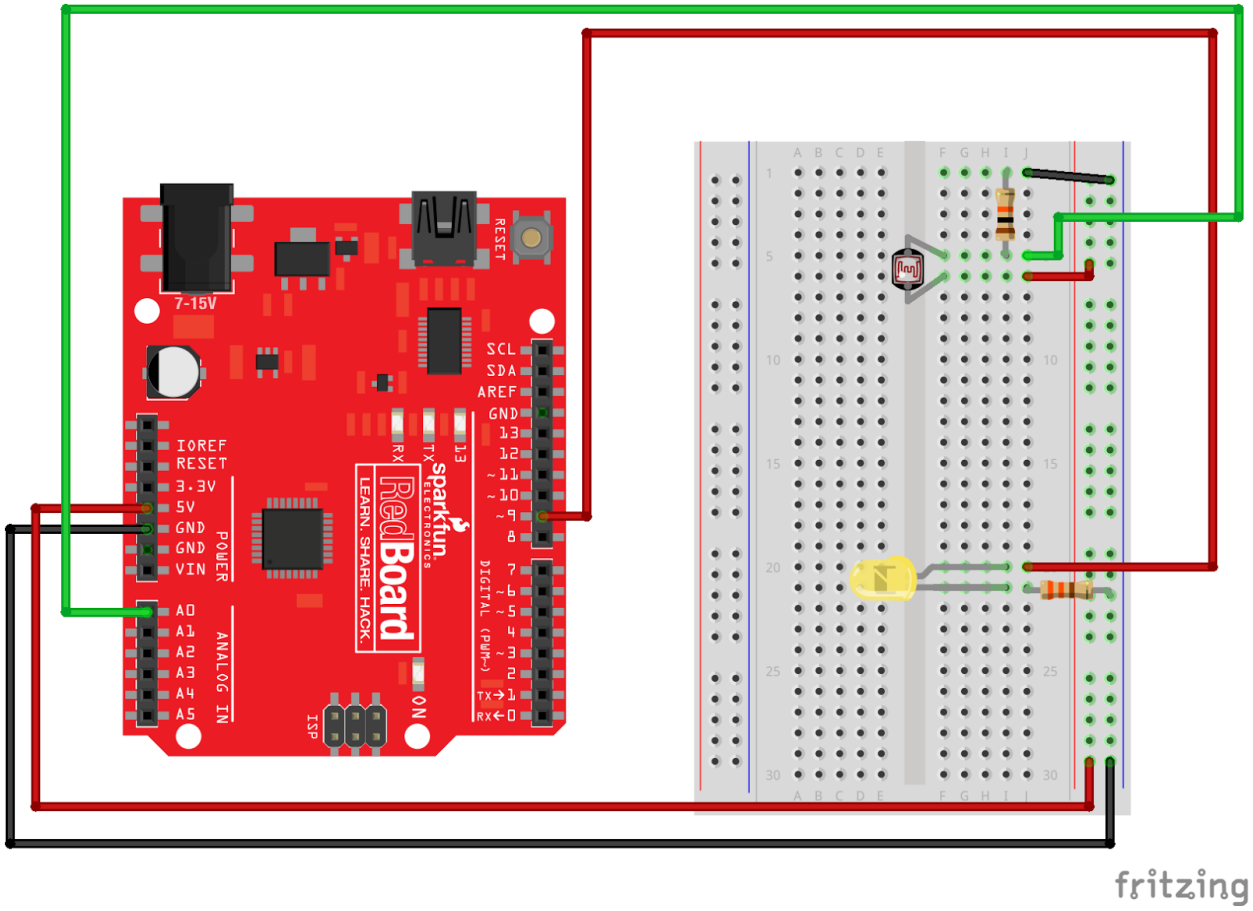
- **digital** - digital means that it can only be one of two states. We generally talk about this as either 0 or 1, true or false, and HIGH or LOW. Using the digitalRead() command, we can see if a pin is either HIGH or LOW.

- **state machine** - a state machine is a way to control the behavior of the microcontroller using a state variable. The loop() is broken down into two parts. The first part is the trigger or event which changes the state. We change the state by setting the state variable to a new number. The second part is the actual state machine. It is a set of nested if() - else if() - else statements that look to see what the state variable is.

Circuit #6 - Photoresistor (Light Detector)

sketch - https://codebender.cc/sketch:77051

This sketch illustrates how to use a sensor to control the brightness of an LED.



Learning objective(s):

1. Voltage divider circuit
2. Serial Monitor
3. range of analogRead()
4. mapping from one range to another (y = mx+b)
5. setting threshold values / calibration
6. **commands / functions to introduce:**
    a. map([val],[fromMin],[fromMax], [toMin], [toMax]) function
    b. constrain([val],[min],[max]) function
    c. compound boolean operators: &&, ||, !
    d. Serial object
        i. Serial.begin();
        ii. Serial.print();
        iii. Serial.println();

Vocabulary / Concepts:

- **Voltage divider -** The photoresistor circuit in this example is a voltage divider circuit. The resistance of the photoresistor varies from a range of <100 Ohms to >10kOhms depending on the amount of light. The analog input of the Arduino is looking for a voltage between 0 and 5V -- so, we use a voltage divider circuit to do this.

- **interpolation / mapping -** If we want to scale and offset a value from one range to another another range, we can use the map() command. This is essentially an application of y = mx+b. If you teach math, you can use the following notation for the map function:

```
y = map(x, x1, x2, y1, y2);  // this returns a y-value for a given x. It puts this on
                             // a line defined by the points (x1, y1) and (x2, y2)
```

  For everyone else, we describe this function as:

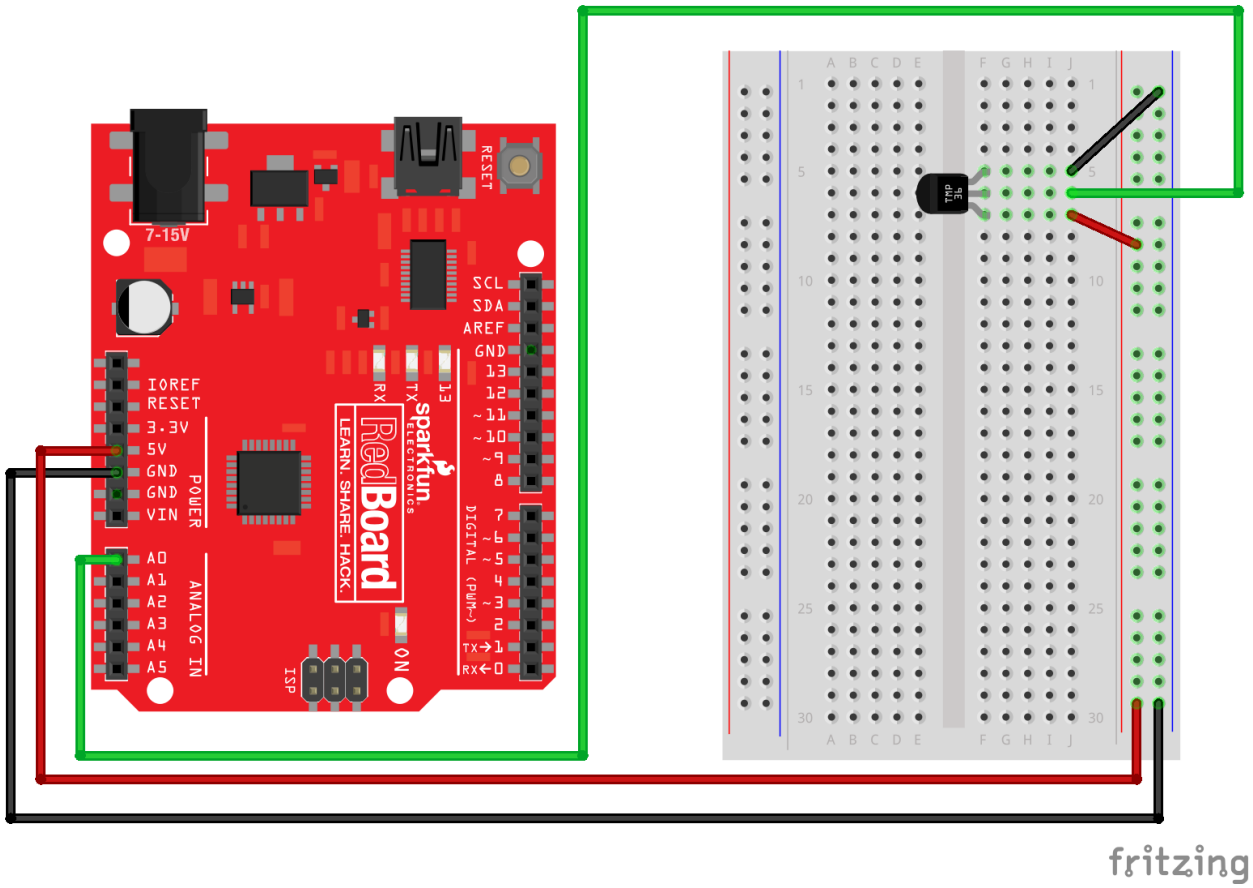
```
newValue = map(oldValue, fromMin, fromMax, toMin, toMax);
```

- **Serial communication** - microcontrollers and other electronic devices often communicate by Serial. This utilizes two separate lines - one for transmit (TX) and another for receive (RX). It must be agreed upon what speed the transmission will be at. Common speeds are 9600 bits/second (bps), 14400, 38400, and 57600. When we use this on the Arduino, you will see the TX and RX lights flash when data is being transmitted.
- **Serial "object"** - we introduce here the Serial object. The Serial object is a way of manipulating data transmitted to and received from another device (usually the computer). The commands all start with the word "Serial."

Circuit #7 - Temperature Sensor (TMP36)

[example code](#) - https://codebender.cc/sketch:77052

In this sketch, we will show you how to interface to a simple temperature sensor, the TMP36. This temperature sensor is linear across temperature from a range of



Learning objective(s):

1. **float** decimal variable type
2. Applying a calibration equation / scaling -- from voltage to temperature (y=mx+b)
3. **commands / functions to introduce:**
    a. float
    b. Writing arithmetic assignments
    c. Writing / calling functions that return a value

Vocabulary / Concepts:

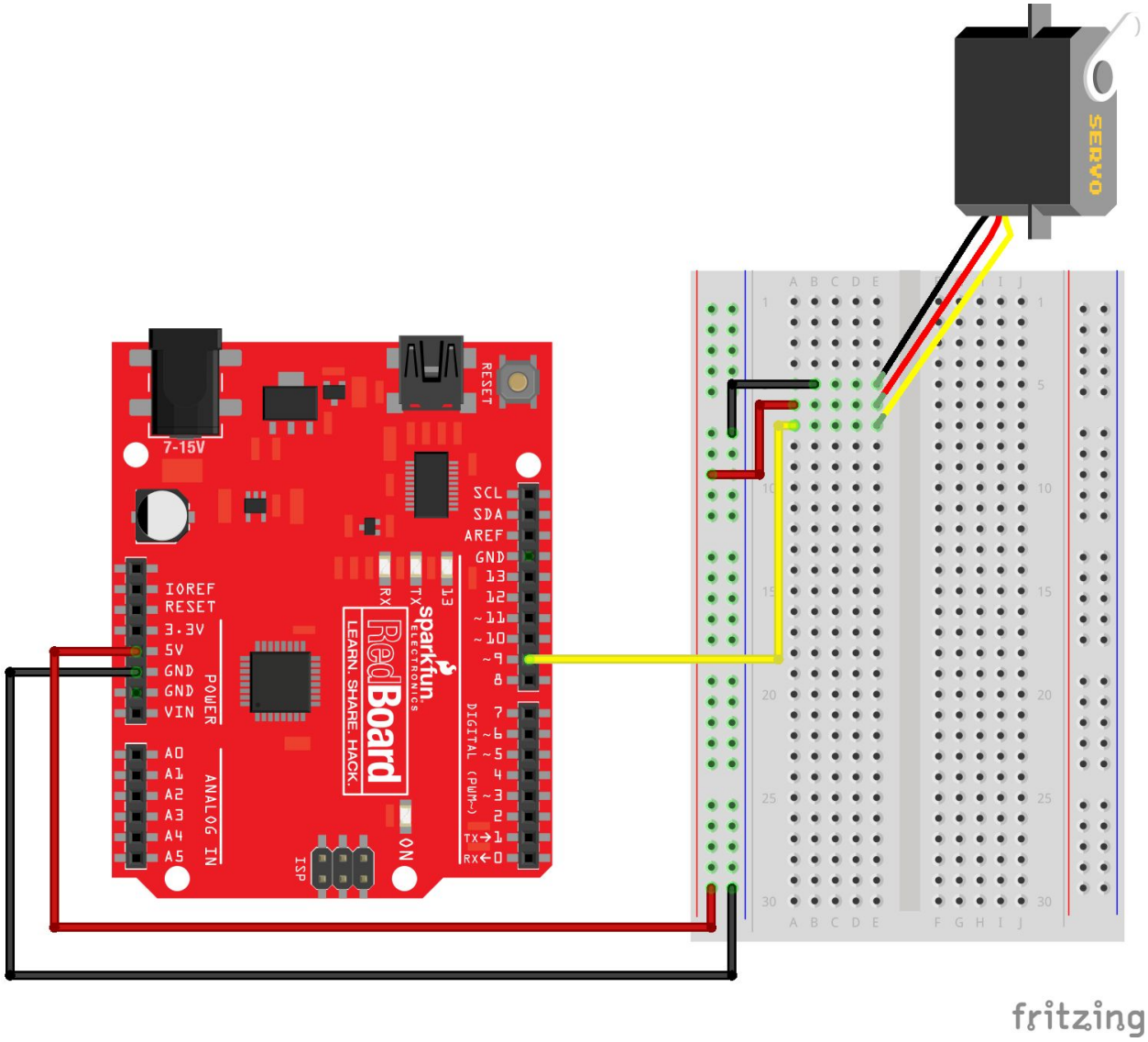
- float

- serial communication
- volts per LSB
- scaling / offset -- y = mx + b
- converting from voltage to temperature.
- TMP36 Sensor - reading datasheets

Circuit #8.1 - Servo Sweep

[example code](#) - https://codebender.cc/sketch:77055

This is the first of two sketches to explore how we can interface to a Servo motor. With many things in Arduino, we use libraries or pre-written code to help. The Servo is a perfect example of this.



Learning objective(s):

1. Understand what a servo motor is and how it works
2. Use and manipulate the servo library & object.
3. Applying a calibration equation / scaling -- from voltage to temperature (y=mx+b)
4. **commands / functions to introduce:**
    a. #include<Servo.h>
    b. Using the Servo object:

i. Servo myServo;
ii. myServo.attach();
iii. myServo.write();

c. for() loop

Vocabulary:

- servo
- object
- PWM

Circuit #8.2 - Serial Servo

sketch - https://codebender.cc/sketch:77053

Learning objective(s):

1. Manipulate input from the Serial Monitor using the Serial.read() command.
2. **commands / functions to introduce:**
   a. Using the Serial object:
      i. Serial.available()
      ii. Serial.read()
      iii. Serial.parseInt()
   b. constrain()

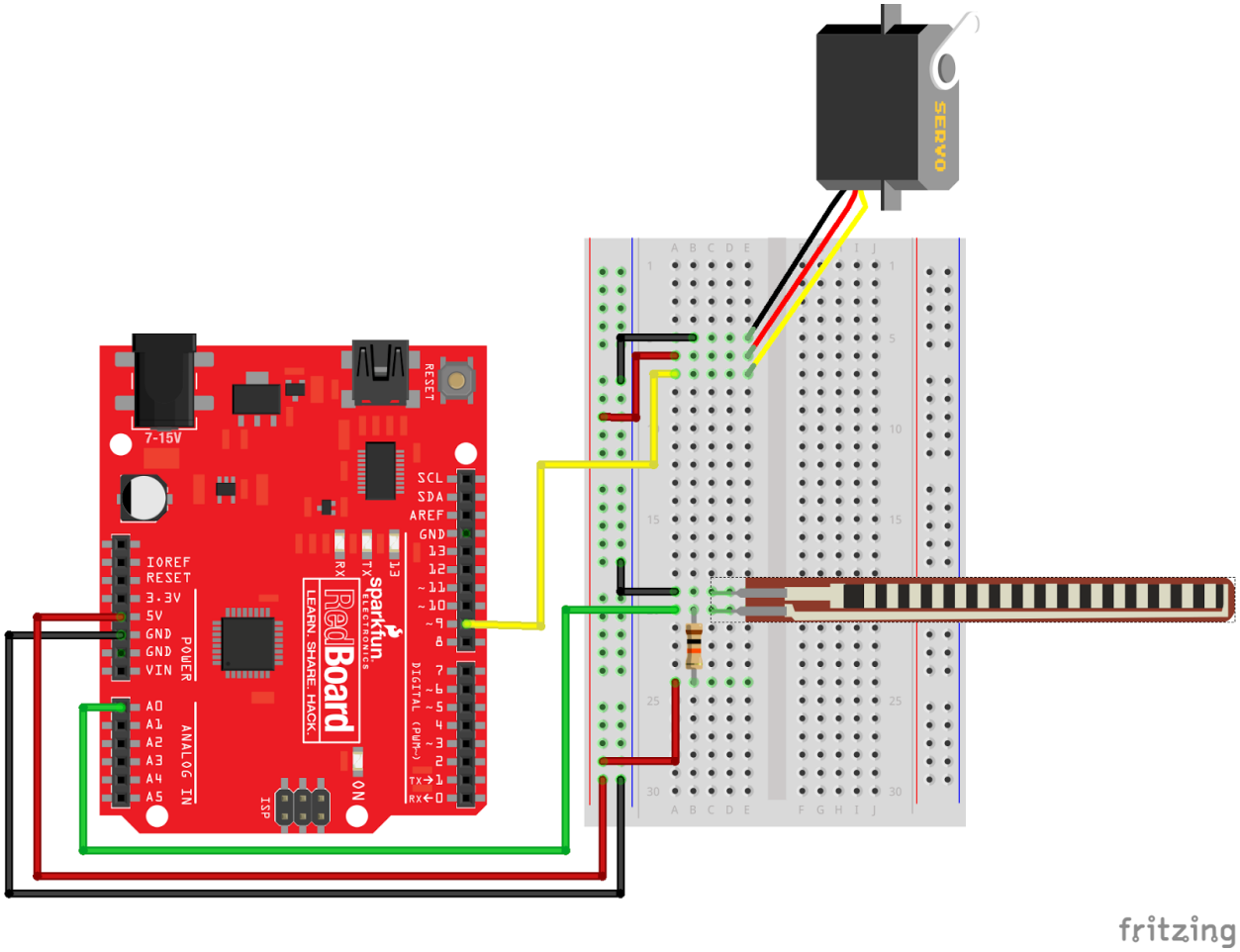
Vocabulary:

- servo
- object

Circuit #9 - Flex Sensor

example code - https://codebender.cc/sketch:77057

This example uses the flex sensor input to control the motion of the Servo motor.



Learning objective(s):

1. Find the range of resistance values of the flex sensor for some nominal range of motion.
2. Apply Ohms law / voltage divider circuit to calculate the theoretical range of input values using a 10k Ohm pull-up or pull-down resistor.
3. Use an input value from the flex sensor to drive a servo.
4. **commands / functions to introduce:**
    a.

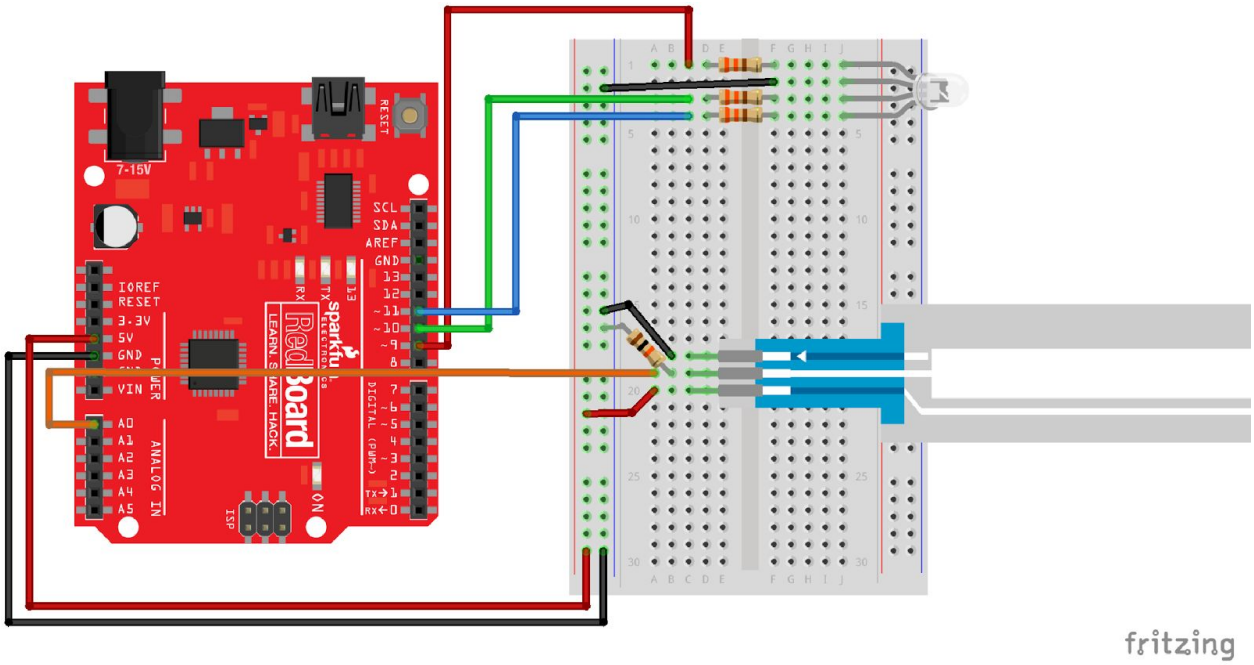
Vocabulary:

- Voltage Divider

Circuit #10 - Soft Potentiometer

example code - https://codebender.cc/sketch:77058

Use the soft potentiometer to change the color of the RGB LED.



The soft potentiometer (soft pot for short) is a neat input device that detects pressure along its length. When you press it down with a finger (it works best on a flat surface), it will change resistance depending on where you're pressing it. You might use it to make a piano or light dimmer; here we're going to use it to control the color of an RGB LED.

The middle pin of the soft potentiometer *floats* when you are not pressing down on the sensor. When it is floating, the voltage will bounce around and give spurious readings to the Arduino. To alleviate this, a pull-down resistor is used so that the input is nominally LOW or 0 V until you press down on the soft potentiometer.

Learning objective(s):

1. Scaling input values (0 to 1023) to control fading between three LEDs.
2. Floating input
3. Pull-down resistor
4. **commands / functions to introduce:**
    a. No new commands / functions introduced in this code example, but these are used:
        i. map([val],[fromMin],[fromMax], [toMin], [toMax]) function
        ii. nested if(), if() else, if() else if() else

iii. boolean comparison operators: ==, >, <, >=, <=

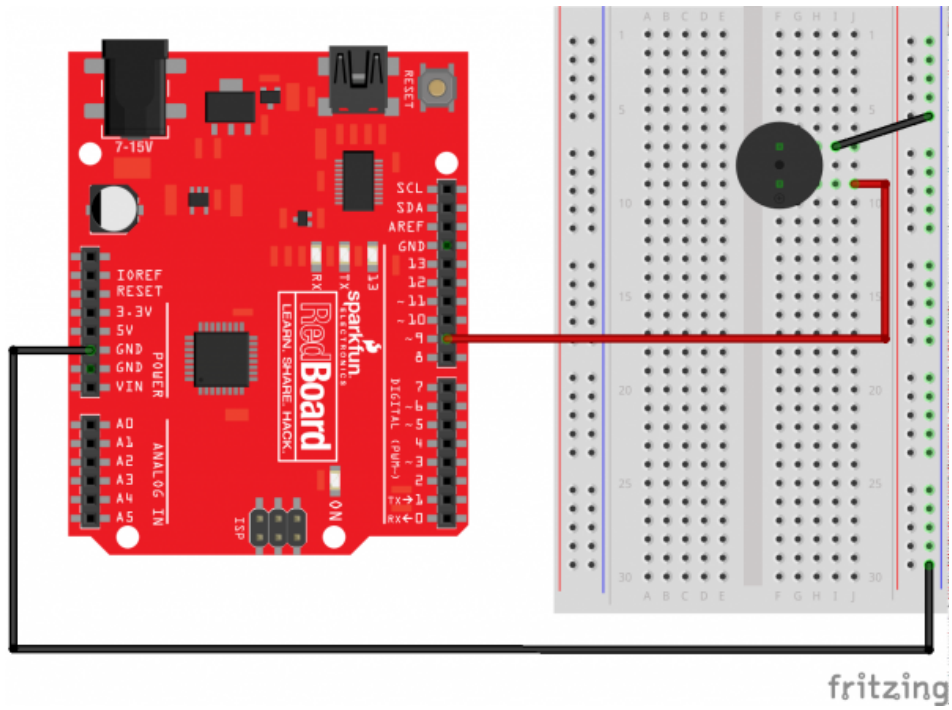
Vocabulary:

- **soft potentiometer** - A soft potentiometer is a 3-pin device also known as a variable resistor. Similar to the knob that we used in circuit #2, the resistance between the center pin and the two outer pins changes depending on where you apply pressure.

Circuit #11 - Buzzer

[example code](https://codebender.cc/sketch:77059) - https://codebender.cc/sketch:77059

This sketch uses the buzzer to play songs. It uses the Arduino's tone([pin], [freq]) function to play notes of a given frequency.



Learning objective(s):

1. Introduce a new variable type called char.
2. Use a char array[].
3. Integrate for() loops to have both notes and durations controlled by an array.
4. **commands / functions to introduce:**
    a. tone([pin], [freq]);
    b. arrays
    c. for() loops

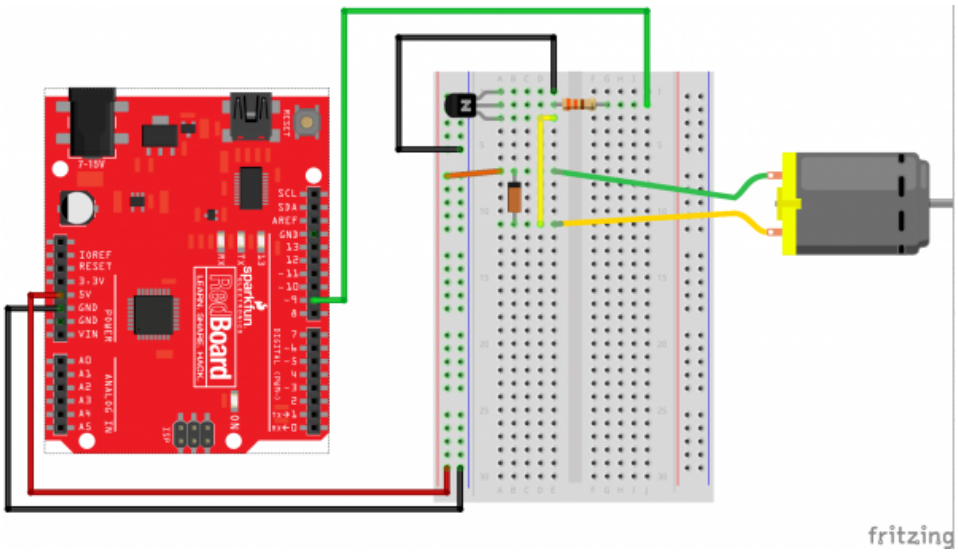
Vocabulary:

- Piezo Element (Buzzer)
- frequency
- notes
- octave
- tone
- array

Circuit #12 - Driving a Motor

[example code](#) - https://codebender.cc/sketch:77060

This example requires that you drive your motor using a switching transistor. The Arduino is only capable of sourcing about 40 mA of current per pin and a motor requires upwards of 150 mA. A transistor is basically a semi-conductor switch (sometimes called a solid-state switch). When a small signal is applied to the Base of a transistor, it turns ON and allows current to flow from the Collector to the Emitter.



Learning objective(s):

1. Understand current / power / current limits of Arduino.
2. Understand how to wire up a transistor to switch higher currents with a "low voltage" signal.
3. Define the use of a transistor.
4. Understand the need for a "fly-back" diode because motors generate Back EMF when they start / stop.
5. **commands / functions to introduce:**
    a. No new commands / functions introduced in this code example.

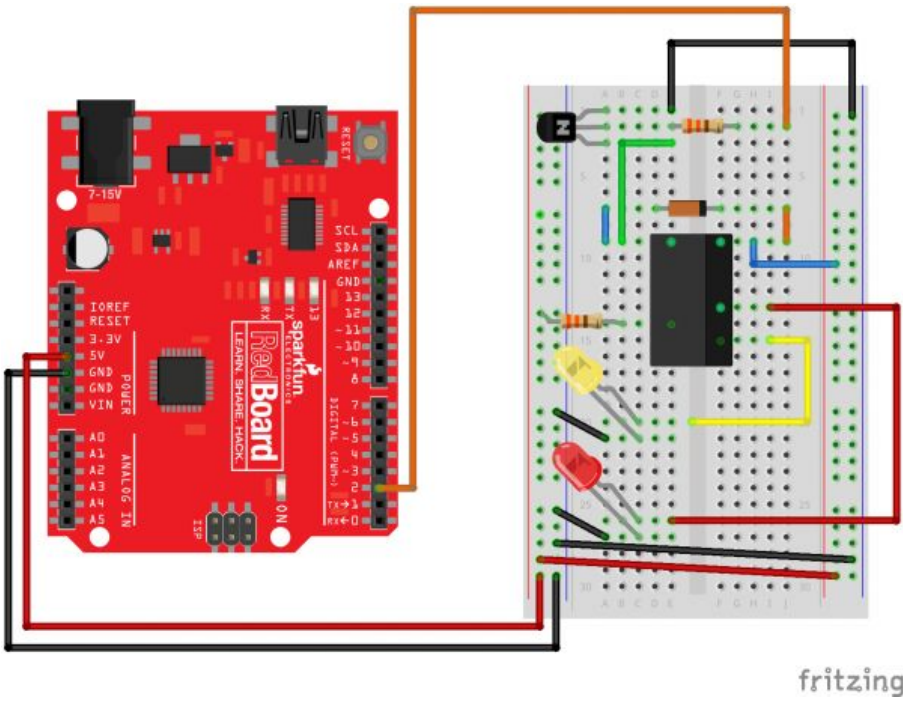
Vocabulary:

- Motor
- Current
- Transistor (Base, Collector, Emitter)
- Fly-back diode

Circuit #13 - Relays

example code - https://codebender.cc/sketch:77061

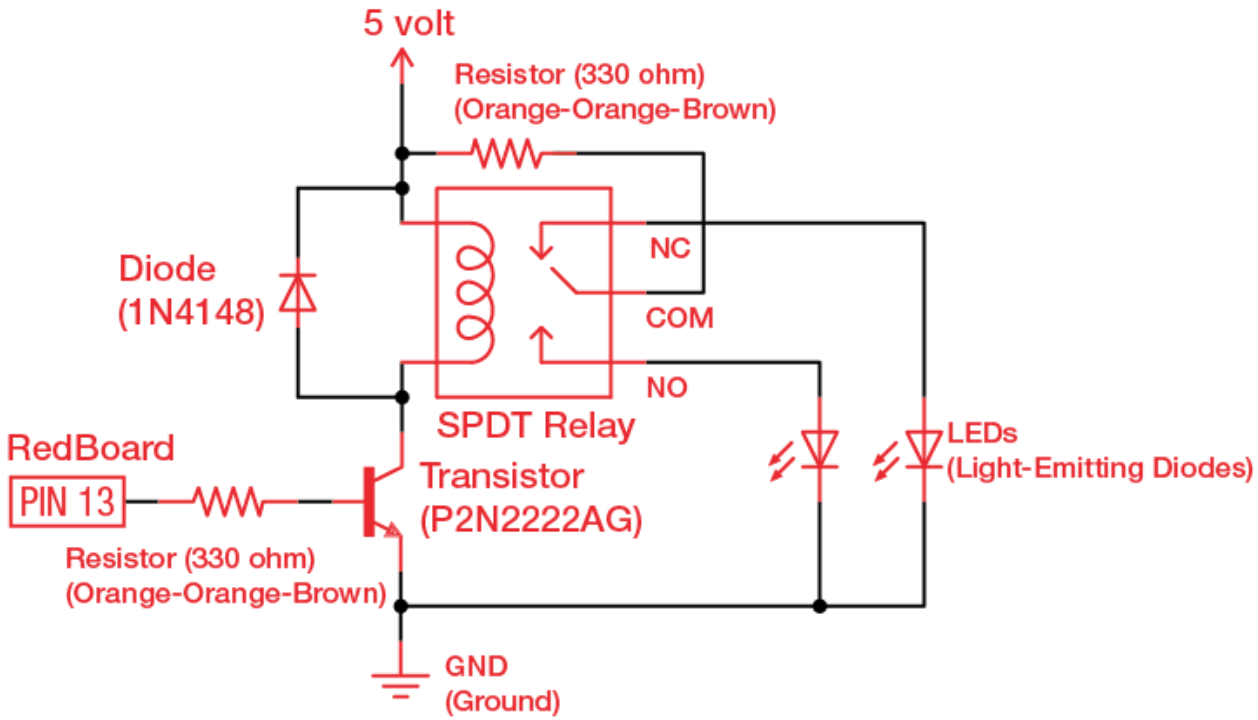
This example demonstrates how to switch HIGH POWER devices using a relay. A relay is a small electromechanical switch that works by exciting an electromagnet that pulls a latch open or close. Because the Arduino can only source about 40 mA of current per pin, we need to use the transistor circuit from Circuit #12 to drive the relay. Relays make a clicking sound when they switch on and off. You sometimes hear these in cars when you turn on the lights or turn on your turning signal.



Circuit Diagram:

The circuit diagram helps explain how this circuit works. This circuit is really similar to the motor circuit you used in example #9. When Pin 13 is HIGH, the transistor is turned ON. This allows current to flow through the coil of the relay. When current flows through the coil, this creates an electromagnet which switches the relay.

When Pin 13 is LOW, the transistor is turned OFF. No current flows through the coil, and with no magnet present, a spring flips the switch back to its normal state. This relays has 5 pins on it. Two pins are used for the coil. The other three pins are labelled: Normally Open (NO), Normally Closed (NC), and Common (COM).



Learning objective(s):

1. Understand current / power / current limits.
2. Complex circuit building - integrating multiple circuits together.
3. **commands / functions to introduce:**
    a. No new commands / functions introduced in this code example.

Vocabulary:

- relay
- electromagnet
- load

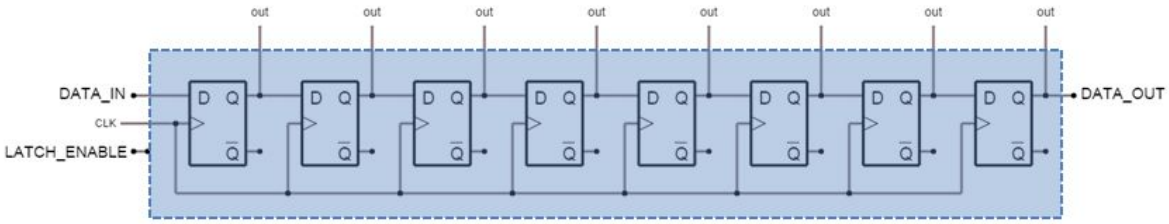
Circuit #14 - Controlling Multiple Outputs -- Shift Register

[example code](#) - https://codebender.cc/sketch:77062

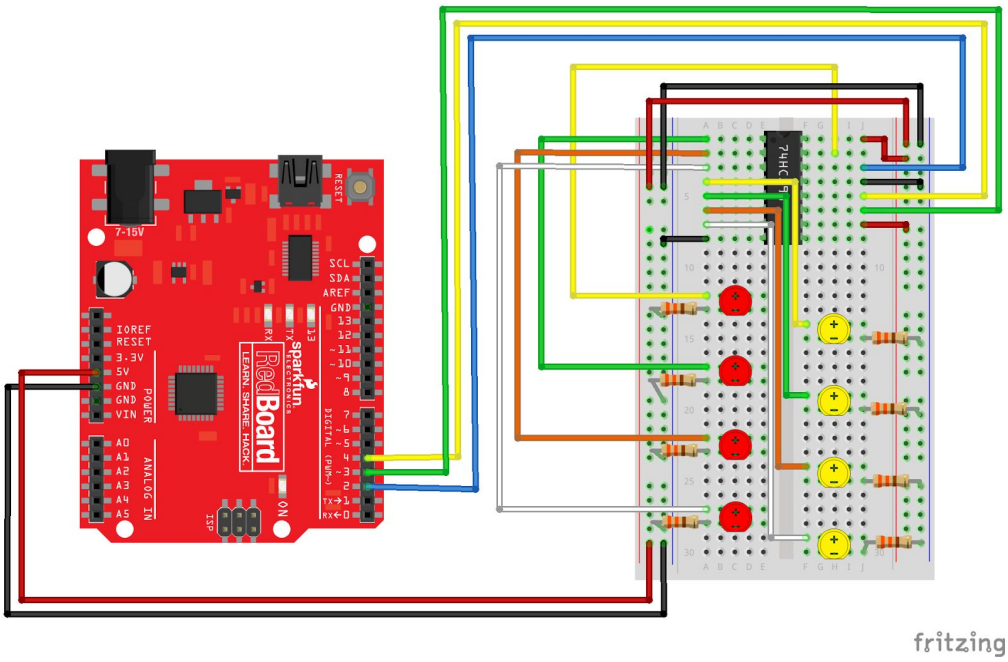
In circuit #4, we controlled 8 LEDs using 8 pins on the Arduino. The Arduino only has 20 GPIO pins. If we wanted to drive more than 20 LEDs, we need a way to do this more efficiently. This circuit introduces students to a device called a shift register.

A **register** is another name for a memory storage device. Each "register" stores a single *bit* of data - either a 1 or a 0. The shift register is like a chain and works by "shifting" the data in one bit at a time. The advantage of this is that we can control any number of LEDs now with just 3 signal pins. Here's a simplified diagram of a shift register. It uses three inputs - **Data In**, **Clock**, and **Latch Enable**. The Clock signal works like a metronome and synchronizes the transfer of data - one bit at a time. To move in 8 bits, the clock signal will go up and down 8 times.

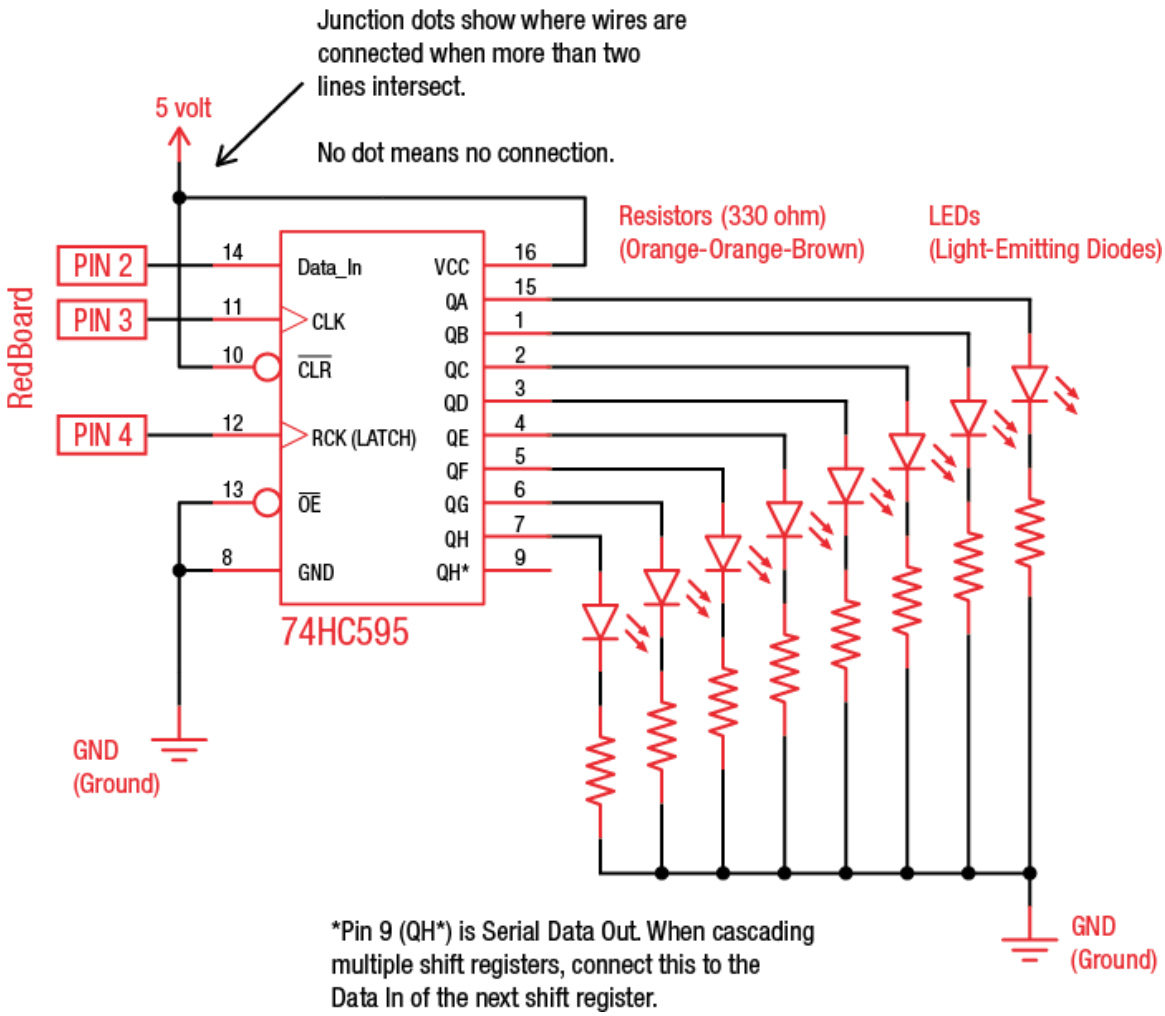
If you wanted to add more than 8 outputs, multiple shift registers can be cascaded together by connecting the **Data Out** of the shift register to the **Data In** of the next shift register. The second shift register will share the same Clock and Latch Enable pins.



Wiring Diagram:



Circuit Diagram (*On the shift register, pin 1 starts in the upper-left corner and the numbers increase as you go counter-clockwise - despite the drawing*)



Learning objective(s):

1. Understanding how a shift register works.
2. Understanding how you can load data serially by "shifting" data in 1 bit at a time.
3. Understand the difference between a bit and a byte.
4. Understanding a "clock" and "latch"
5. **commands / functions to introduce:**
    a. bitWrite()
    b. shiftOut()

Vocabulary:

- Shift Register
- Clock
- Latch
- Cascade

Circuit #15 - Liquid Crystal Display (LCD)

example code - https://codebender.cc/sketch:77063

Adding a display to an Arduino project is always a fun thing for students. Standard liquid crystal displays (LCD) have 16 pins exposed for: power, register select, data (8 pins), backlight, and contrast control. There is an LCD driver chip (Hitachi HD44780U) that is at the heart of the display. This LCD can display up to 16 characters x 2 lines. In our example, we use just 6 I/O lines on the Arduino for: register select (RS), enable, and a 4-bit data line.

Register select (RS): Used to select between sending the LCD "instructions" and "data".

Enable: Allows the Arduino to control whether or not the LCD is updated.

Data: (Note: the 4-bit control of the LCD only uses 4 pins on the Arduino for data, but is slower. Students may want to increase this to 8 data lines if they are seeing flickering.

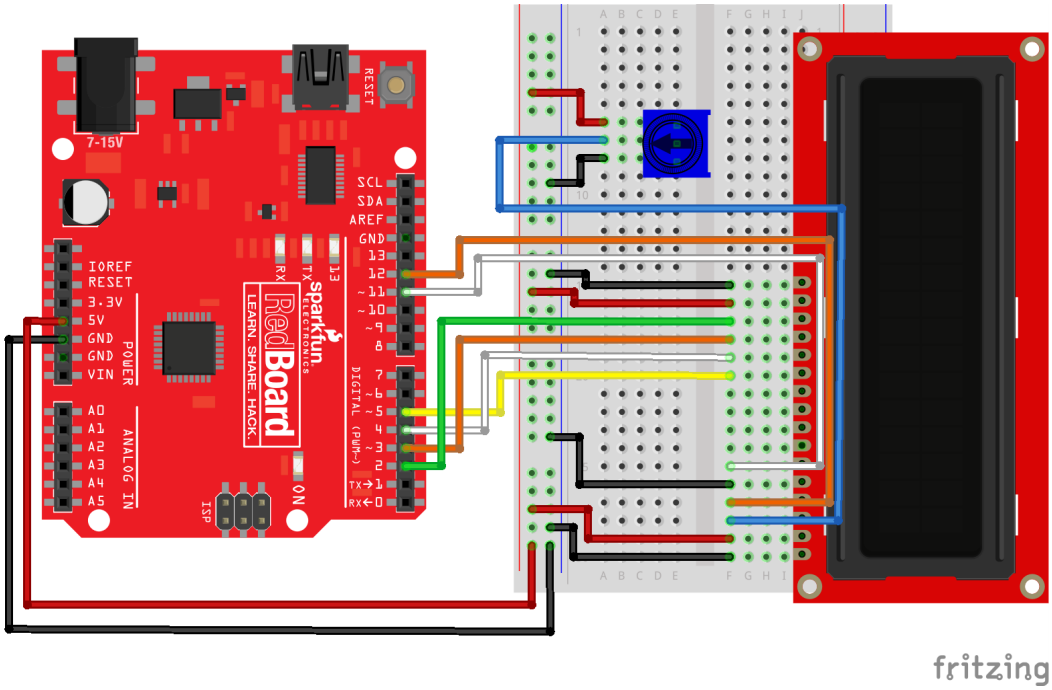
This example sketch uses the Arduino LiquidCrystal.h library. Go to: https://www.arduino.cc/en/Reference/LiquidCrystal for more information on other things you can do with this display.

The LCD display is a handy way to add a display to your Arduino project. Rather than using the Serial monitor, you can now display information directly to the LCD display. This will enable you to separate your Arduino project from your computer.

Challenge your students to combine Circuit #7 with this one to display the temperature of the room directly to the LCD display. Or, combine both Circuit #7 and Circuit #6 to display both temperature and ambient light conditions.

Advanced:
Each character is a 5 x 8 dot matrix bitmap. Students can explore using the CreateChar() method of the LiquidCrystal library. Students can create up to 8 of their own unique "glyphs" using a 5 x 8 matrix and display these to the LCD.

Wiring Diagram:



Learning objective(s):

1. Wiring and organization
2. Using the LiquidCrystal library
3. Reading the Arduino [library reference] to discover other features & tricks of interfacing the LCD.
4. **commands / functions to introduce:**
   a. `LiquidCrystal lcd(RS, Enable, d4, d5, d6, d7);`
      `LiquidCrystal lcd(12, 11, 5, 4, 3, 2);`
      `// initializing the LCD object called "lcd". Additional LCDs`
      `// could be used using other pins on the Arduino. A second`
      `// lcd would need to be named "lcd2" or something different.`

   b. Using methods of the LiquidCrystal library:
      `.begin(16, 2);`
      `.clear();`
      `.print("Hello World!");`
      `.setCursor(col, row);`

Vocabulary:

- Liquid Crystal
- contrast
- dot matrix
- glyph

This LCD is a parallel input display. Pins on the LCD are numbered 1 through 16. Data is transferred to the display across 4 data lines labelled 11 - 14. A full byte (8-bits) of data is transferred to the display using two 4-bit data transfers.

Because we are only writing data to the LCD, the R/W (Read/Write Select) pin can be tied the GND. Communication to the LCD is accomplished using just 6 wires from the Arduino (not including Power and Ground). The LCD library handles all of the communication and signalling necessary.

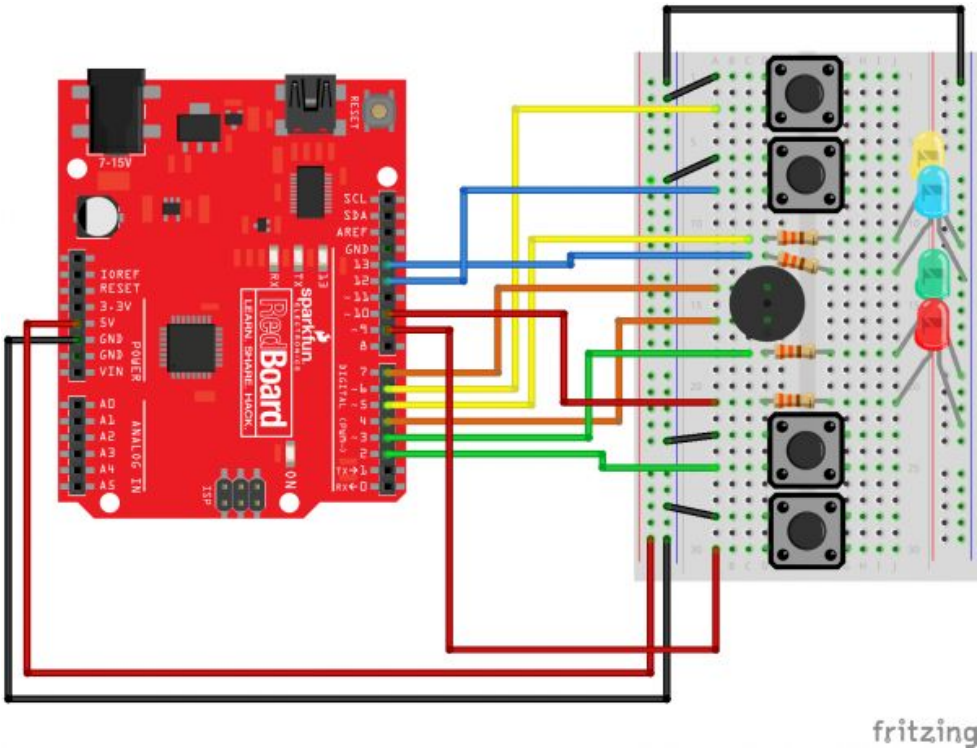
Circuit #16 - Simon Game

[example code](https://codebender.cc/sketch:77064) - https://codebender.cc/sketch:77064

This final project incorporates 4 LEDs, 4 buttons, and the buzzer to re-create the classic Simon game popularized by Milton Bradley. This is just one illustration of how we can use something simple, 4 LEDs, 4 buttons, and a buzzer -- to create a fun game. We challenge students to think about what makes a game? Encourage students to think about the elements of a game:

1. Space
2. Goals
3. Components
4. Mechanics
5. Rules

Bothe the wiring and the code example here is fairly advanced. For most students, we encourage them to modify and manipulate the existing code -- rather than try to write something from scratch. Possible ideas include changing the tones or adding a 5th button and a 5th LED. Another challenge is to have students move this circuit off of the breadboard and onto cardboard or other enclosure. This teaches students how to read circuit diagrams and abstract these to a unique design.



Learning objective(s):

1. Integrating multiple piece of code together using a state machine
2. Breaking up a complex piece of code into individual functions
3. Parsing and making sense of code
4. Use of bit-masking (See the function `setLEDs(byte leds)`)
5. **commands / functions to introduce:**
    a. Bitwise operators: &, |, !
    b. custom functions to "chunk" or compartmentalize existing code.

Vocabulary:

- Bitwise AND (&), OR (|), NOT (!)