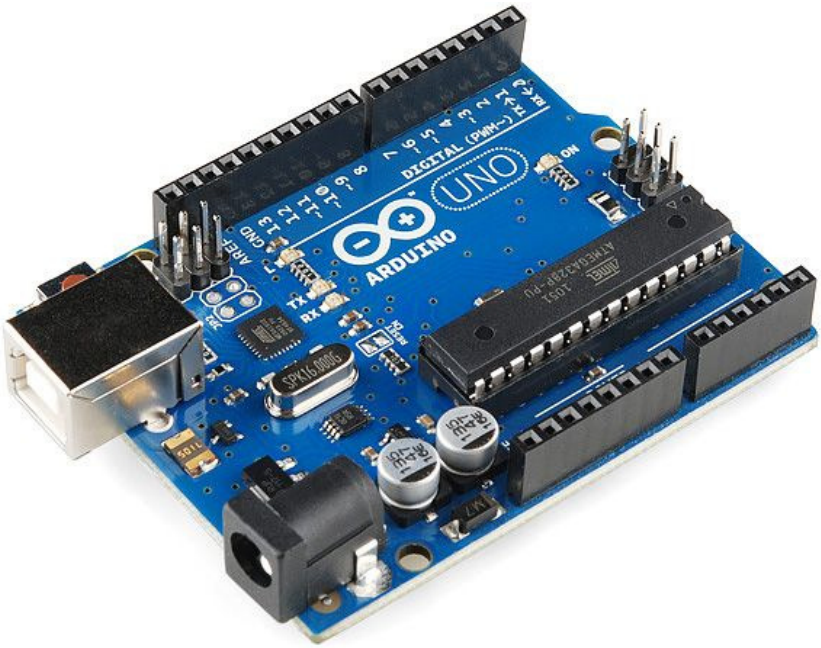
# *Introduction to Microcontrollers*



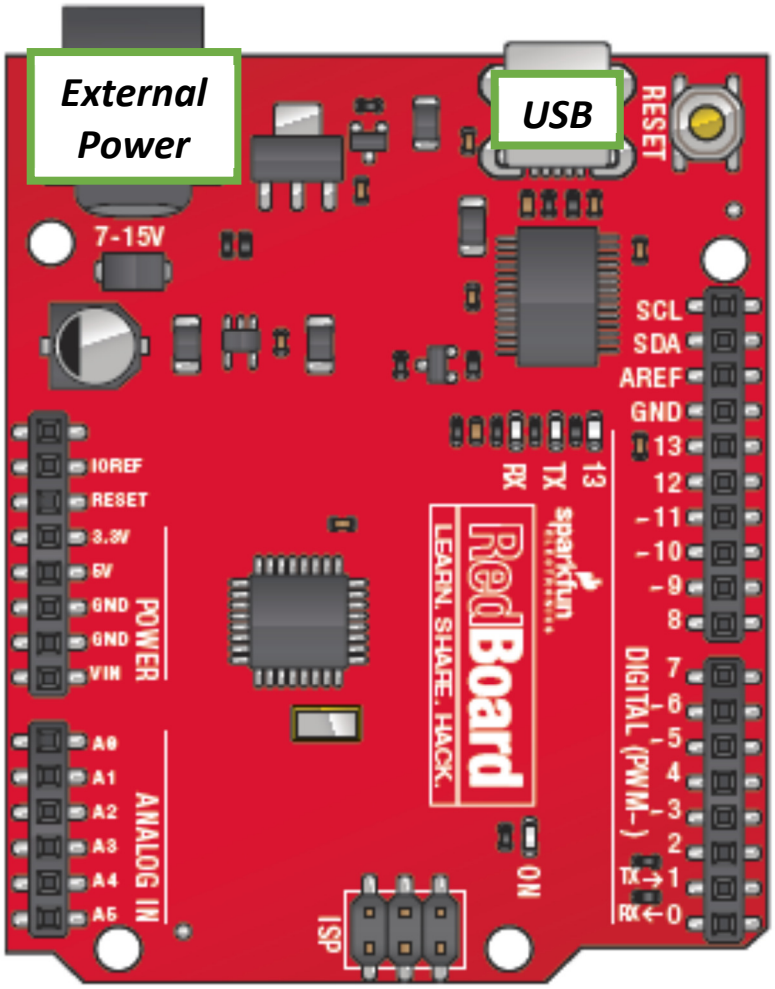

Programmable digital computers have become the standard for replacing complex circuitry and digital logic chips by allowing users to utilize a simple programming language to tell electronic components how to behave. The major advantage of these miniature "computers" is that they can be programmed to read digital or analog inputs from the environment, and then output digital signals based on the inputs as instructed by the programmer.

The most common version of these microcontrollers has been the "Arduino" and its variants. The basic Arduino Uno is an 8-bit computer about the size of a credit card which can process many different inputs and outputs using a very versatile and easy to learn programming environment.

The experiments in this book are designed to introduce the concepts of the Arduino physical computing system and provide students with a basic understanding for using microcontrollers with projects.

# *The Sparkfun RedBoard*

The microcontrollers that we will be using are clones of the Arduino Uno manufactured by Sparkfun and called "RedBoards". They use the same programming code and have all of the same features of and Uno, with a few additions.



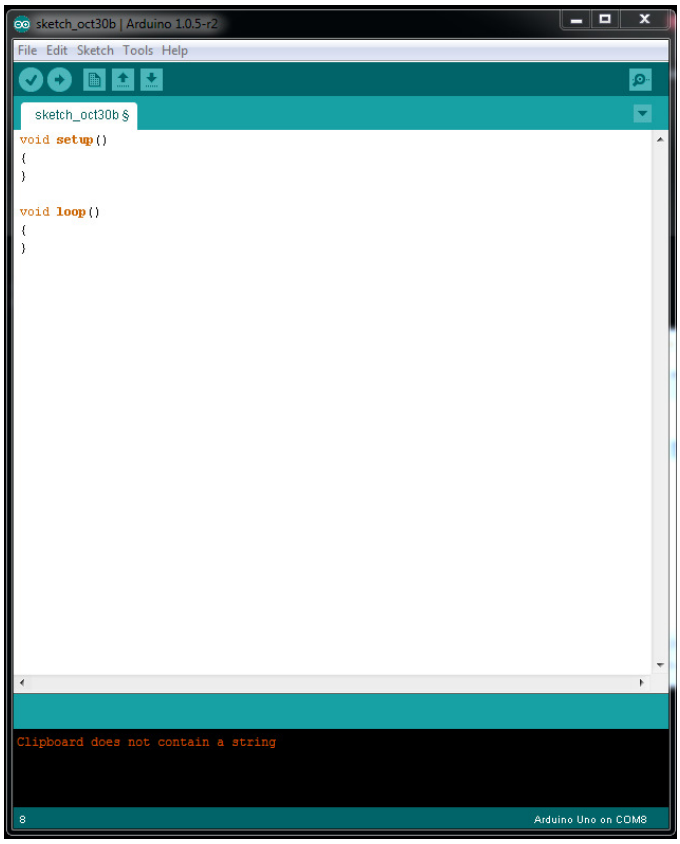

Power to the boards can be provided using a USB cable connected to a computer, or an external power supply if necessary. Programming the board is done through a USB connection. At the top right of the board is a "Reset" button that can be used to restart a running program on the board.

The pins on the left side of the board labelled "Power" are used to provide power to circuits connected to the board. The most commonly used pins are 5V and GND, providing positive and negative power connections.

On the lower left side of the board are 6 "Analog In" pins. These are used as inputs only, and can read a voltage measurement and assign it a value between 0 (0V) and 1023 (5V). As a voltage reading changes, they assign an appropriate value which can then be used by the board, thus allowing the board to convert analog signals from sensors and convert them to digital signals to be read by the computer.

Along the right side of the board are 14 "Digital" pins that can serve as either inputs or outputs, depending on how they are declared in your program. Digital signals can only be one of two values, referred to as "HIGH" (5V) and "LOW" (0V), which basically correlate to being "ON" or "OFF". Some digital pins are also marked with a tilde (~) symbol which designates them as pins capable of "Pulse Width Modulation (PWM)". PWM is a means of simulating an analog output by switching the pin between HIGH and LOW very quickly, and by varying the speed of the switching they appear as if they are outputting an analog signal.

# *The Arduino IDE*




Programs in the Arduino IDE system are often referred to as “sketches”, and when you begin a new sketch you will often see the two essential pieces of any Arduino code: the “void setup” and “void loop” commands. You will write your code in between the curly brackets that follow these commands, and every sketch you write must have both of these commands.

To connect the programming window to your RedBoard, you can use the “Tools” menu to select both the board and the communications port that it is connected to. We will treat our board as an Arduino Uno and then select the available COM port that shows up in the menu. Our connection is shown on the bottom right of the window.


When your sketch is completed you can check it for errors before loading it to the board by clicking the “Verify” button on the top menu.

Once your sketch has been verified and no errors are found, you can upload it to the board by clicking the “Upload” button on the top menu.

If you get an error while uploading your sketch to the board, make sure that your board is plugged into the computer and you have chosen the correct COM port and board, then try again.

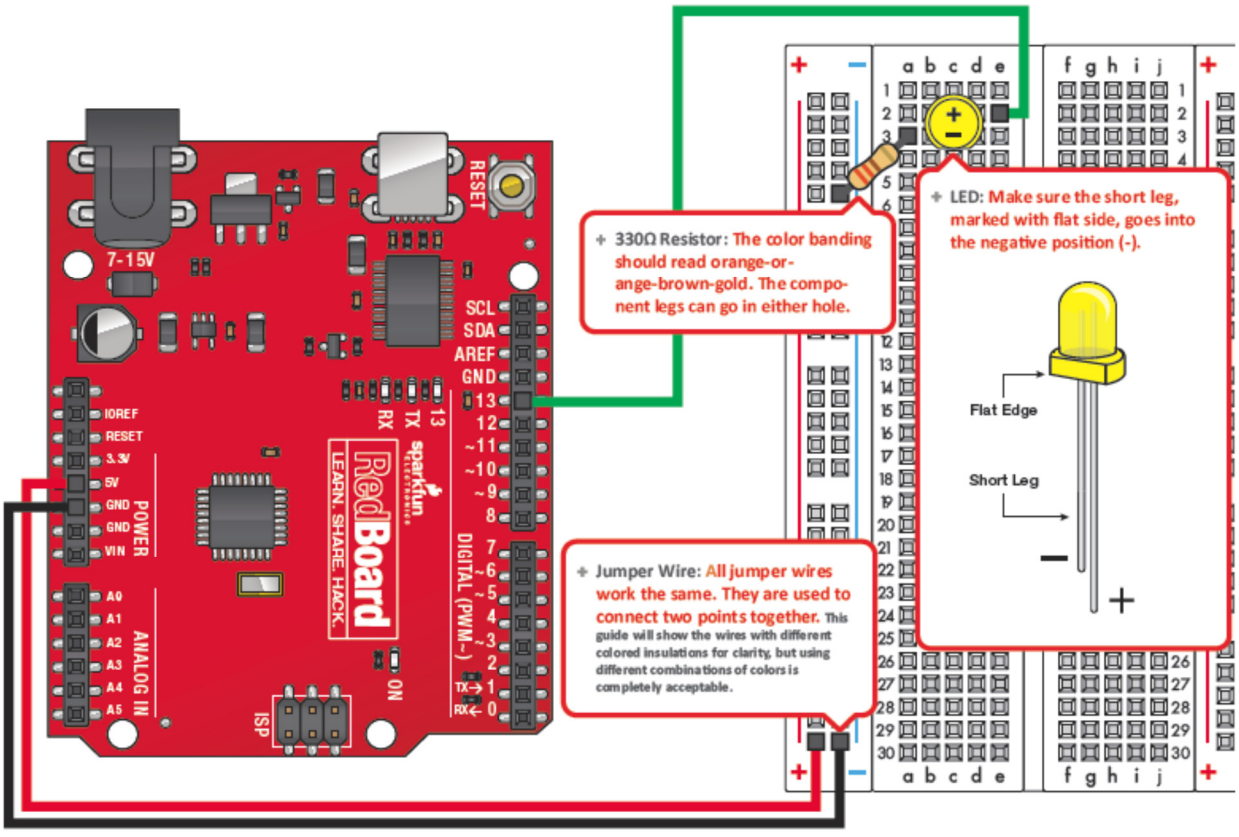
When uploading a program, sometimes the communications port can become “clogged” and will report an error uploading to the board. This can sometimes be avoided by pressing the “reset” button on the board before uploading the sketch, or you can simply upload the sketch again until it correctly communicates with the board.

# *Experiment 1*

In this experiment we will introduce the Arduino programming environment, which is often referred to as the “Arduino IDE” (Integrated Development Environment). We will set up a very simple circuit with an LED connected to a digital output pin on the RedBoard, then instruct the board to make the LED blink on and off at a specified rate.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE THE LED (ANY COLOR) ON THE BREADBOARD AND CONNECT THE NEGATIVE END TO GROUND USING A 330Ω RESISTOR.
- USE A JUMPER WIRE TO CONNECT DIGITAL PIN 13 ON THE REDBOARD TO THE POSITIVE END OF THE LED.



## PROGRAM

- THE “SETUP” PART OF OUR CODE WILL RUN FIRST AND WILL ONLY RUN ONCE. THIS IS WHERE WE TELL THE BOARD HOW TO SET ITSELF UP FOR THE INSTRUCTIONS TO FOLLOW.
- IN THIS CASE, WE WILL NEED TO TELL IT WHICH PIN WE’LL BE USING AND HOW WE’LL BE USING IT
    - THERE ARE 14 DIGITAL PINS ON OUR BOARD, EACH OF WHICH CAN BE EITHER AN INPUT OR AN OUTPUT.
    - BECAUSE WE’LL BE SENDING A SIGNAL FROM THE BOARD TO OUR LED, WE NEED TO DESIGNATE OUR DIGITAL PIN AS AN OUTPUT.
- THE pinMode FUNCTION TAKES TWO VALUES, THE NUMBER OF THE PIN WE’RE USING AND THEN EITHER “INPUT” OR “OUTPUT”.
    - IN OUR CASE WE ARE USING PIN 13 AS AN OUTPUT.
- REMEMBER TO END EACH COMPLETE STATEMENT WITH A SEMICOLON

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

## PROGRAM

- THE “LOOP” PART OF OUR CODE WILL RUN AFTER SETUP AND WILL CONTINUE RUNNING UNTIL STOPPED. THIS IS WHERE THE ACTUAL PROGRAM INSTRUCTIONS ARE LOCATED AND IS USUALLY THE BULK OF THE PROGRAM
- WE WANT TO WRITE A COMMAND TO DIGITAL PIN 13, SO WE USE THE digitalWrite FUNCTION.
    - DIGITAL PINS CAN ONLY BE EITHER “HIGH” (5V) OR “LOW” (0V), SO TO TURN OUR LED ON, WE WRITE A HIGH COMMAND TO PIN 13.
    - ONCE OUR LED IS ONE, WE WILL WRITE A LOW COMMAND TO PIN 13 IN ORDER TO TURN IT OFF AGAIN.
- IN ORDER TO SLOW DOWN THE PROGRAM, WE INCLUDE A delay FUNCTION BETWEEN THE WRITE COMMANDS.
    - THE NUMBER IN PARENTHESES IS THE NUMBER OF MILLISECONDS THE PROGRAM WILL WAIT UNTIL EXECUTING THE NEXT LINE OF CODE.

```
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
}
```

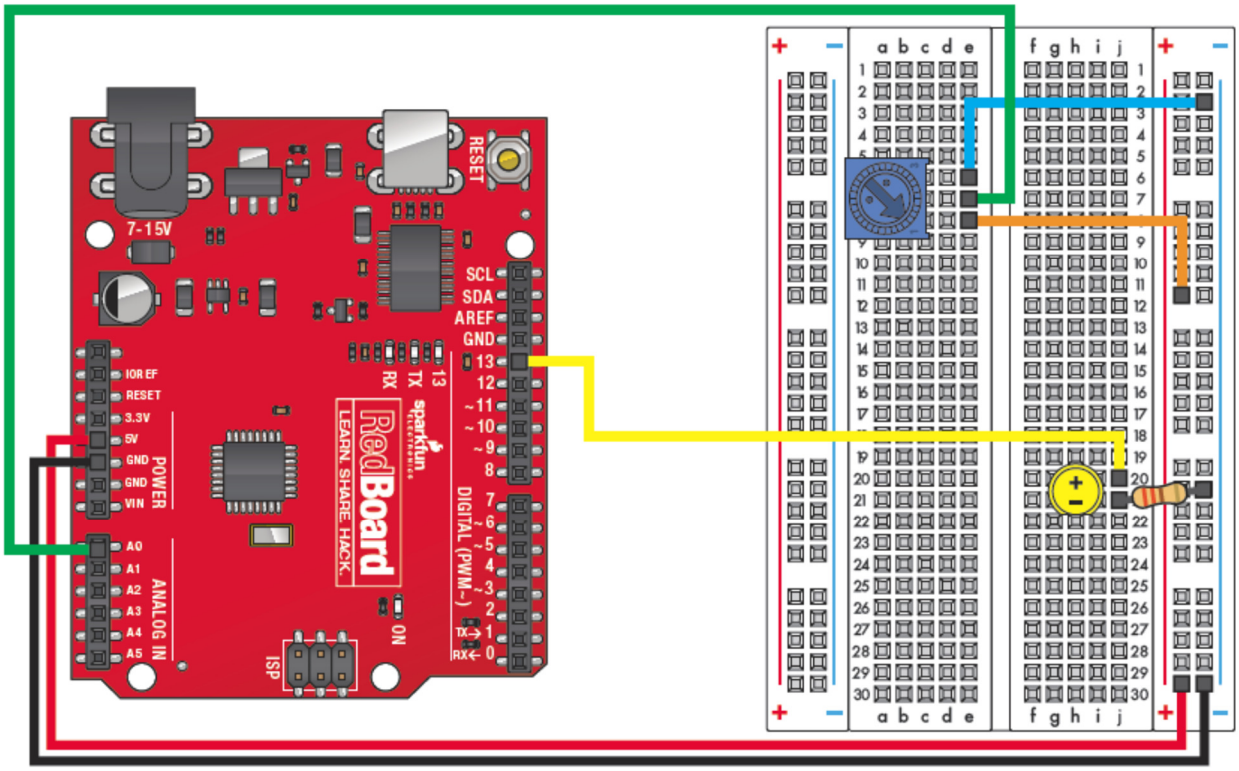
```
void setup()
{
  pinMode(13, OUTPUT);
}

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
}
```

# *Experiment 2*

In this experiment we will introduce an analog input to our circuit and use it to control the rate at which an LED blinks on and off. By reading the voltage values across a potentiometer we will see how the RedBoard translates an analog signal into a digital one.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE THE LED (ANY COLOR) ON THE BREADBOARD AND CONNECT THE NEGATIVE END TO GROUND USING A 330Ω RESISTOR.
- PLACE A 2KΩ POTENTIOMETER ON THE BREADBOARD AND CONNECT THE OUTSIDE PINS TO POSITIVE AND GROUND.
- CONNECT DIGITAL PIN 13 ON THE REDBOARD TO THE POSITIVE END OF THE LED.
- CONNECT ANALOG PIN 0 ON THE REDBOARD TO THE CENTER PIN OF THE POTENTIOMETER.



## PROGRAM

- WE CAN INTRODUCE "VARIABLES" TO DEFINE TERMS THAT WILL BE USED IN THE CODE. THIS OFTEN PROVIDES AN EASIER WAY TO TRACK LATER INSTRUCTIONS.
  - HERE WE DESCRIBE TWO "INTEGER" (int) VARIABLES. THE TERM INTEGER SIMPLY REFLECTS THAT THEY ARE WHOLE NUMBERS.
  - WE SET THE TERMS sensorPin EQUAL TO 0, AND ledPin EQUAL TO 13.
  - BECAUSE THESE ARE DECALRED BEFORE THE setup PART OF THE SKETCH THEY ARE KNOWN AS "GLOBAL" VARIABLES, AND CAN BE USED THROUGHOUT THE SKETCH.
- WE AGAIN SET PIN 13 AS AN OUTPUT USING THE pinMode FUNCTION, HOWEVER, WE CAN NOW USE THE TERM ledPin SINCE WE DEFINED IT AS EQUAL TO 13.
  - WE DO NOT NEED TO SET OUR ANALOG PIN AS AN INPUT, BECAUSE ANALOG PINS CAN ONLY BE INPUTS.

```
int sensorPin = 0;

int ledPin = 13;

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

## PROGRAM

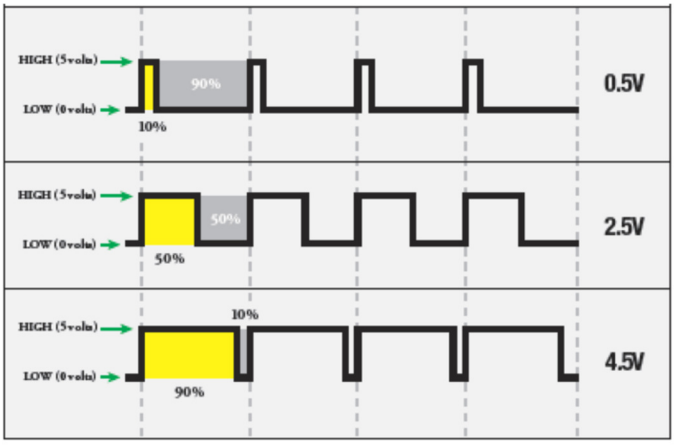
- INSIDE OUR LOOP WE WILL DECLARE A “LOCAL” VARIABLE, WHICH ONLY IS TRUE INSIDE THIS PART OF THE CODE.
    - WE’LL CALL THIS int VALUE sensorValue.
- WE WANT TO READ INFO FROM AN ANALOG PIN, SO WE USE THE analogRead FUNCTION TO READ THE VOLTAGE FROM OUR sensorPin (DEFINED EARLIER).
- TO TURN OUR LED ON, WE WRITE A HIGH COMMAND TO ledPin.
- TO TURN OUR LED OFF, WE WRITE A LOW COMMAND TO ledPin.
- THE INCLUDED delay FUNCTION WILL DEPEND ON THE VALUE BEING READ BY THE ANALOG PIN FROM THE POTENTIOMETER.
    - THE DELAY TIME WILL BE A NUMBER BETWEEN 0 (0V) AND 1023 (5V), DEPENDING ON THE VOLTAGE READ BY THE ANALOG PIN.

```
void loop()
{

  int sensorValue;

  sensorValue = analogRead(sensorPin);

  digitalWrite(ledPin, HIGH);      // Turn the LED on

  delay(sensorValue);              // Pause for sensorValue
                                   // milliseconds

  digitalWrite(ledPin, LOW);       // Turn the LED off

  delay(sensorValue);              // Pause for sensorValue
                                   // milliseconds

}
```

```
int sensorPin = 0;

int ledPin = 13;

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{

  int sensorValue;

  sensorValue = analogRead(sensorPin);

  digitalWrite(ledPin, HIGH);     // Turn the LED on

  delay(sensorValue);             // Pause for sensorValue
                                  // milliseconds

  digitalWrite(ledPin, LOW);      // Turn the LED off

  delay(sensorValue);             // Pause for sensorValue
                                  // milliseconds

}
```

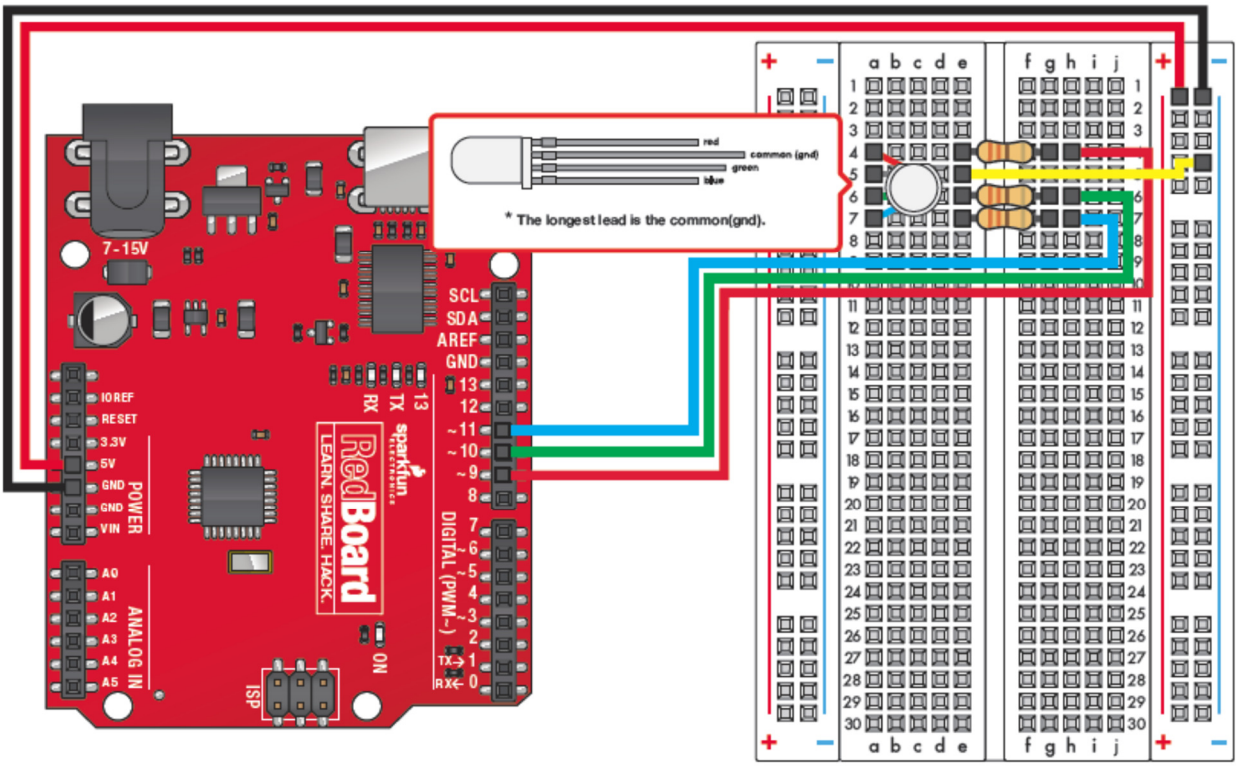
# *Experiment 3*

This experiment utilizes the Pulse Width Modulation (PWM) aspect of certain digital output pins to simulate an analog output. PWM emits timed pulses oscillating between HIGH and LOW, for example by changing between 50% HIGH and 50% LOW it appears to give a 50% signal (i.e. 2.5V). There are 256 ($2^8$) different possible timings, so a 50% signal would mean setting it to value 128.




## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE THE RGB LED ON THE BREADBOARD AND CONNECT THE NEGATIVE PIN (LONGEST PIN) TO GROUND.
- CONNECT THE OTHER THREE PINS OF THE RGB LED TO 330Ω RESISTORS.
- THROUGH THE RESISTORS, CONNECT THE LED'S RED, GREEN, AND BLUE PINS TO DIGITAL PINS 9, 10, AND 11, RESPECTIVELY, ON THE BOARD.



## PROGRAM

- WE WILL SET UP GLOBAL VARIABLES FOR THE PINS WE WILL BE USING.
    - THE VARIABLES WILL AGAIN BE INTEGERS (int), BUT WE WILL ALSO DECLARE THEM TO REMAIN CONSTANT (const) THROUGHOUT THE SKETCH.
    - BECAUSE WE'LL BE SENDING A SIGNAL FROM THE BOARD TO OUR LED, WE NEED TO DESIGNATE OUR DIGITAL PIN AS AN OUTPUT.
- UNDER setup WE WILL AGAIN USE THE pinMode FUNCTION TO SET EACH OF OUR PINS AS AN OUTPUT.
- INSIDE OUR loop WE WILL SETUP FOUR MORE VARIABLES AS INTEGERS, BUT WE WILL NOT GIVE THEM VALUES.
    - THE VALUES OF THESE VARIABLES WILL BE DETERMINED BY EQUATIONS IN THE SKETCH.

```
const int RED_PIN = 9;
const int GREEN_PIN = 10;
const int BLUE_PIN = 11;

void setup()
{
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}

void loop()
{
  int x;
  int redIntensity;
  int greenIntensity;
  int blueIntensity;
```

## PROGRAM

- NEXT WE WILL SET UP A "FOR LOOP". FOR LOOPS RUN OVER AND OVER UNTIL A CERTAIN CONDITION IS FINALLY MET.
- EVERY for() FUNCTION TAKES THREE STATEMENTS:
  - SOMETHING TO DO BEFORE STARTING (SET x = 0)
  - A TEST TO PERFORM (IF x < 768, KEEP LOOPING)
  - SOMETHING TO DO AFTER EACH LOOP (INCREASE THE VALUE OF X BY 1)
    - x++ IS A SHORTCUT FOR x=x+1.
- OUR FOR LOOP WILL START AT X=0, THEN ADD 1 TO X EVERY LOOP, AND KEEP LOOPING UNTIL X=768.
- AFTER THE for() FUNCTION, EVERYTHING INSIDE THE CURLY BRACKETS IS PART OF THE FOR LOOP.

```
for (x = 0; x < 768; x++)
{
  if (x <= 255)          // zone 1
  {
    redIntensity = 255 - x;    // red goes from on to off
    greenIntensity = x;        // green goes from off to on
    blueIntensity = 0;         // blue is always off
  }
  else if (x <= 511)     // zone 2
  {
    redIntensity = 0;                  // red is always off
    greenIntensity = 255 - (x - 256); // green on to off
    blueIntensity = (x - 256);         // blue off to on
  }
  else // color >= 512       // zone 3
  {
    redIntensity = (x - 512);          // red off to on
    greenIntensity = 0;                // green is always off
    blueIntensity = 255 - (x - 512);  // blue on to off
  }

  analogWrite(RED_PIN, redIntensity);
  analogWrite(BLUE_PIN, blueIntensity);
  analogWrite(GREEN_PIN, greenIntensity);
  delay(10);   // Delay for 10 ms (1/100th of a second)
}
}
```

## PROGRAM

- INSIDE OUR FOR LOOP WE WILL CREATE SEVERAL "IF…ELSE" FUNCTIONS. THESE FUNCTIONS WILL EXECUTE ONLY IF A CERTAIN CONDITION IS MET.
- AN if() FUNCTION TAKES ONE TEST TO PERFORM, AND THEN WILL RUN THE FOLLOWING CODE IN CURLY BRACKETS IF THAT TEST IT TRUE. IF THAT TEST IS FALSE IT WILL MOVE ON TO THE NEXT PART OF THE SKETCH
    - OUR TEST WILL BE IF x <= 255. IF THAT IS TRUE THEN IT WILL RUN THE CODE SETTING THE RED, GREEN, AND BLUE INTENSITY VARIABLES AS DESCRIBED. OTHERWISE IT WILL MOVE ON.
- AN else if() FUNCTION MEANS IF THE FIRST TEST IS FALSE, BUT A SECOND TEST IS TRUE, THEN RUN THE FOLLOWING CODE.
- FINALLY, AN else FUNCTION RUNS IF ALL OF THE PREVIOUS TESTS ARE FALSE.

```
  for (x = 0; x < 768; x++)
  {
    if (x <= 255)          // zone 1
    {
      redIntensity = 255 - x;    // red goes from on to off
      greenIntensity = x;        // green goes from off to on
      blueIntensity = 0;         // blue is always off
    }
    else if (x <= 511)     // zone 2
    {
      redIntensity = 0;                  // red is always off
      greenIntensity = 255 - (x - 256); // green on to off
      blueIntensity = (x - 256);         // blue off to on
    }
    else // color >= 512       // zone 3
    {
      redIntensity = (x - 512);          // red off to on
      greenIntensity = 0;                // green is always off
      blueIntensity = 255 - (x - 512);  // blue on to off
    }

    analogWrite(RED_PIN, redIntensity);
    analogWrite(BLUE_PIN, blueIntensity);
    analogWrite(GREEN_PIN, greenIntensity);
    delay(10);   // Delay for 10 ms (1/100th of a second)
  }
}
```

## PROGRAM

- FINALLY WE CAN WRITE OUR VALUES TO THE DIGITAL PWM PINS.
- IN ORDER TO SIGNIFY THAT WE WANT TO USE PWM INSTEAD OF A STANDARD DIGITAL SIGNAL, WE USE THE analogWrite FUNCTION.
    - MUCH LIKE digitalWrite, analogWrite NEEDS A PIN NUMBER (RED_PIN) AND A VALUE (redIntensity).
    - WITH analogWrite OUR VALUE NEEDS TO BE AN INTEGER BETWEEN 0 AND 255 SO IT CAN SET THE PULSE WIDTH ACCORDINGLY.
- LAST WE WILL delay FOR 10MS BETWEEN EACH LOOP.

```
  for (x = 0; x < 768; x++)
  {
    if (x <= 255)          // zone 1
    {
      redIntensity = 255 - x;    // red goes from on to off
      greenIntensity = x;        // green goes from off to on
      blueIntensity = 0;         // blue is always off
    }
    else if (x <= 511)     // zone 2
    {
      redIntensity = 0;                  // red is always off
      greenIntensity = 255 - (x - 256); // green on to off
      blueIntensity = (x - 256);         // blue off to on
    }
    else // color >= 512       // zone 3
    {
      redIntensity = (x - 512);          // red off to on
      greenIntensity = 0;                // green is always off
      blueIntensity = 255 - (x - 512);  // blue on to off
    }

    analogWrite(RED_PIN, redIntensity);
    analogWrite(BLUE_PIN, blueIntensity);
    analogWrite(GREEN_PIN, greenIntensity);
    delay(10);   // Delay for 10 ms (1/100th of a second)
  }
}
```

```
const int RED_PIN = 9;
const int GREEN_PIN = 10;
const int BLUE_PIN = 11;

void setup()
{
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
}
void loop()
{
  int x;
  int redIntensity;
  int greenIntensity;
  int blueIntensity;

  for (x = 0; x < 768; x++)
  {
    if (x <= 255)          // zone 1
    {
      redIntensity = 255 - x;    // red goes from on to off
      greenIntensity = x;        // green goes from off to on
      blueIntensity = 0;         // blue is always off
    }
    else if (x <= 511)     // zone 2
    {
      redIntensity = 0;                  // red is always off
      greenIntensity = 255 - (x - 256); // green on to off
      blueIntensity = (x - 256);         // blue off to on
    }
    else // color >= 512       // zone 3
    {
      redIntensity = (x - 512);          // red off to on
      greenIntensity = 0;                // green is always off
      blueIntensity = 255 - (x - 512);  // blue on to off
    }

    analogWrite(RED_PIN, redIntensity);
    analogWrite(BLUE_PIN, blueIntensity);
    analogWrite(GREEN_PIN, greenIntensity);
    delay(10);   // Delay for 10 ms (1/100th of a second)
  }
}
```

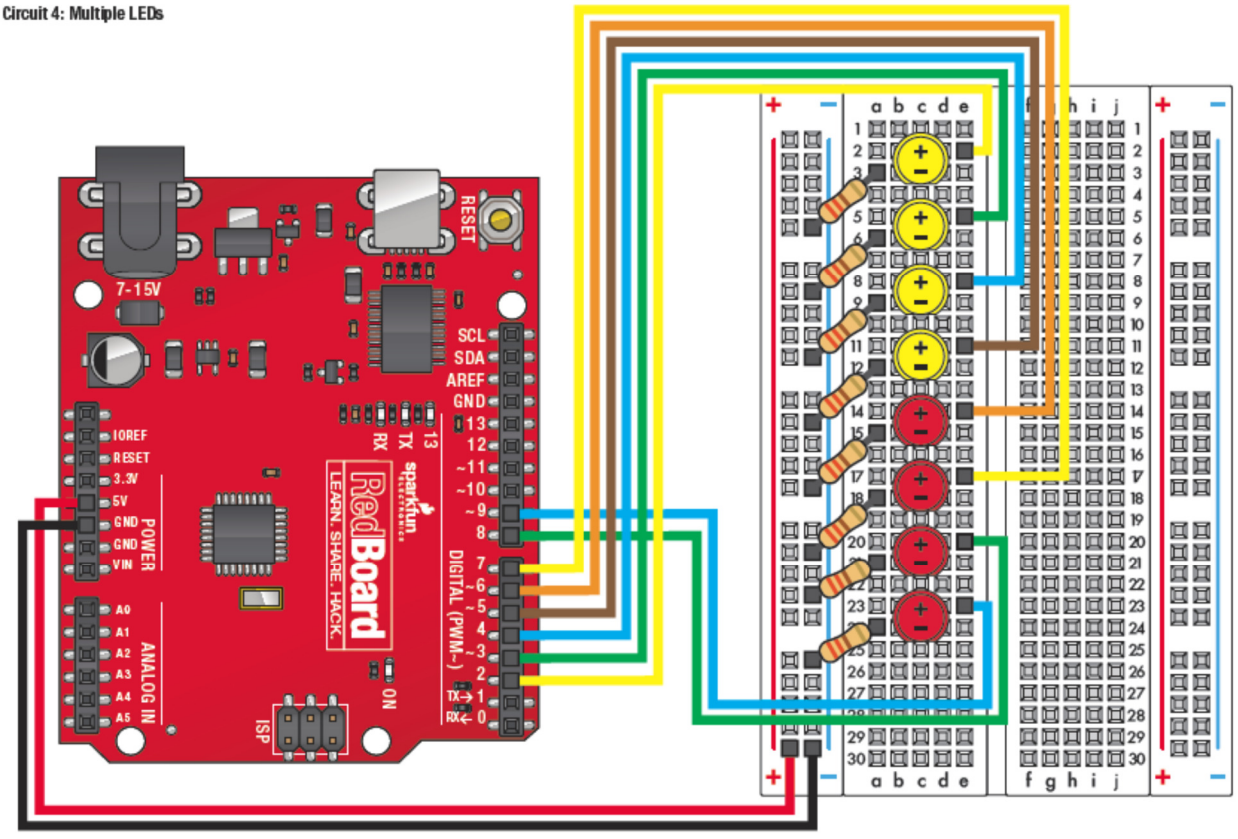
# *Experiment 4*

In this experiment we will light several LEDs in specific patterns using an “array” to determine which one’s should be lit. Arrays save a lot of typing as they allow us to hold many values in a single line of code and then refer back to each value by its “index” number. We’ll also introduce the ability to call up programs from outside our loop command, which is useful if you want to easily choose from several different programs to run

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- SET UP 8 LEDS ON THE BREADBOARD AND CONNECT THEIR NEGATIVE SIDE TO GROUND USING 330Ω RESISTORS.
- USE JUMPER WIRES TO CONNECT THE LED’S POSITIVE PINS TO DIGITAL PINS 2, 3, 4, 5, 6, 7, 8, AND 9, IN ORDER FROM TOP TO BOTTOM, ON THE BOARD.

**Circuit 4: Multiple LEDs**



## PROGRAM

- WE WILL SET UP GLOBAL VARIABLES FOR THE PINS WE WILL BE USING.
    - THE VARIABLES WILL BE INTEGERS (int), BUT WE WILL LIST THEM AS AN "ARRAY" OF VALUES INSTEAD OF LISTING ALL EIGHT INDIVIDUALLY.
- AS SHOWN, OUR ARRAY ledPins[] CONSISTS OF THE STRING OF NUMBERS SEPARATED BY COMMAS BETWEEN CURLY BRACKETS.
- ONCE THE ARRAY IS DEFINED WE CAN REFER TO EACH NUMBER WITHIN IT BY ITS "INDEX", OR THE NUMBER OF ITS POSITION INSIDE THE ARRAY.
    - IN OUR ARRAY, THE NUMBER "2" IS INDEX 0 (THE FIRST ENTRY), "3" IS INDEX 1, "4" IS INDEX 2, AND SO ON.
- WE CAN USE THE INDEX IN OUR setup BY CREATING A FOR LOOP INSTEAD OF TYPING OUT EVERY SINGLE pinMode ENTRY.
    - THE FOR LOOP FIRST SETS THE VARIABLE index = 0, THEN STATES TO RUN THE LOOP AS LONG AS index <= 7, AND AFTER EACH LOOP INCREASE THE VALUE OF INDEX BY ONE (index++).
    - INSIDE THE FOR LOOP WE SET EACH LED PIN AS AN OUTPUT USING pinMode.

```
int ledPins[] = {2,3,4,5,6,7,8,9};

void setup()
{
  int index;

  for(index = 0; index <= 7; index++)
  {
    pinMode(ledPins[index],OUTPUT);
  }
}
```

## PROGRAM

- IN THIS EXPERIMENT, WE WILL USE OUR loop STRUCTURE TO CALL UP ANOTHER PROGRAM FROM SOMEWHERE ELSE IN THE SKETCH BY SIMPLY TYPING THE NAME OF OUR PROGRAM (oneAfterAnotherLoop()).
- WE CAN CREATE OUR PROGRAM ANYWHERE BY NAMING IT WITH THE void COMMAND (void oneAfterAnotherLoop()).
- INSIDE OUR PROGRAM WE WILL USE TWO FOR LOOPS TO LIGHT UP THE LEDS IN ORDER OF THEIR INDEX WITHIN THE ARRAY WE DEFINED EARLIER. for() FUNCTION TAKES THREE STATEMENTS:
    - OUR FIRST LOOP COUNTS UP FROM INDEX 0 TO 7, TURNING ON EACH LED WITH A HIGH COMMAND AND A delay.
    - OUR SECOND LOOP COUNTS DOWN FROM INDEX 7 TO 0, TURNING OFF EACH LED WITH A LOW COMMAND AND A delay.
        - NOTE THAT index-- IS THE SAME AS index++, BUT SUBTRACTING 1 INSTEAD OF ADDING 1.

```
void loop()
{
  oneAfterAnotherLoop();
}
```

////////////////////////////////////////////////////////////////////////

```
void oneAfterAnotherLoop()
{
  int index;
  int delayTime = 100;

  for(index = 0; index <= 7; index++)
  {
    digitalWrite(ledPins[index], HIGH);
    delay(delayTime);
  }
  for(index = 7; index >= 0; index--)
  {
    digitalWrite(ledPins[index], LOW);
    delay(delayTime);
  }
}
```

```
int ledPins[] = {2,3,4,5,6,7,8,9};

void setup()
{
  int index;

  for(index = 0; index <= 7; index++)
  {
    pinMode(ledPins[index],OUTPUT);
  }
}

void loop()
{
  oneAfterAnotherLoop();
}
```

////////////////////////////////////////////////////////////////////

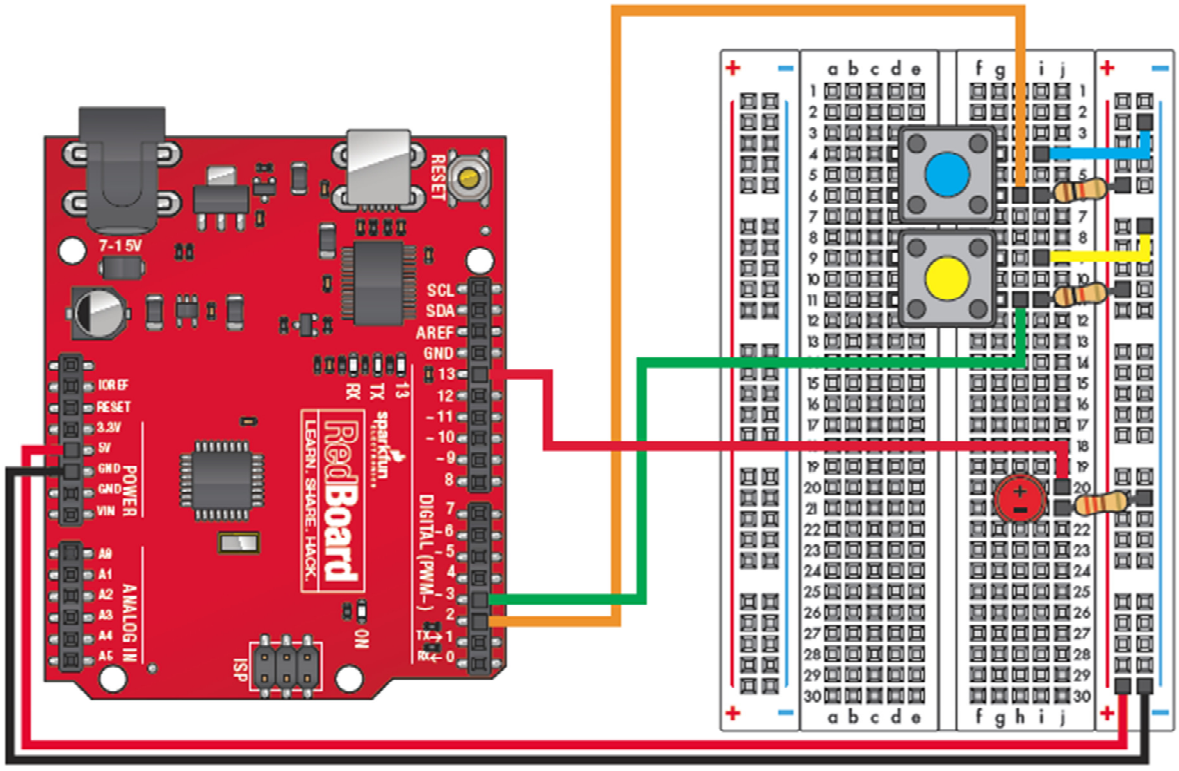
```
void oneAfterAnotherLoop()
{
  int index;
  int delayTime = 100;

  for(index = 0; index <= 7; index++)
  {
    digitalWrite(ledPins[index], HIGH);
    delay(delayTime);
  }
  for(index = 7; index >= 0; index--)
  {
    digitalWrite(ledPins[index], LOW);
    delay(delayTime);
  }
}
```

# *Experiment 5*

This experiment will work by reading the values of digital pins as HIGH or LOW and then reacting to their status. It will also introduce the basic Boolean logic commands for the Arduino IDE. Boolean logic operators are commonly used in all logical applications, but are especially important in computer programming.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE 2 BUTTONS AND AN LED ON THE BREADBOARD. PLACE THE BUTTONS SO THEY STRADDLE THE CENTER BARRIER AS SHOWN IN THE DIAGRAM SO THAT WE DON'T CONFUSE THE CONNECTED AND UNCONNECTED TERMINALS.
- CONNECT ONE TERMINAL OF EACH BUTTON TO GROUND AND THE OTHER TO POSITIVE USING A 1KΩ RESISTOR.
    - THESE ARE CALLED "PULL-UP RESISTORS" AND KEEP THE VOLTAGE FROM "FLOATING" WHEN WE ARE TRYING TO MEASURE IT.
- CONNECT THE NEGATIVE SIDE OF THE LED TO GROUND USING A 330Ω RESISTOR.
- CONNECT DIGITAL PINS 2 AND 3 TO THE POSITIVE TERMINAL OF THE BUTTONS, AND DIGITAL PIN 13 TO THE POSITIVE SIDE OF THE LED.



## PROGRAM

- WE WILL SET UP GLOBAL VARIABLES FOR THE PINS WE WILL BE USING.
    - OUR THREE PIN VARIABLES WILL BE INTEGER CONSTANTS (const int).
- UNDER setup WE CAN SET UP OUR DIGITAL PINS USING pinMode. IN THIS CASE OUR TWO BUTTON PINS WILL EACH BE AN INPUT, WHILE THE LED PIN WILL BE AN OUTPUT.

```
const int button1Pin = 2;
const int button2Pin = 3;
const int ledPin =  13;


void setup()
{
  pinMode(button1Pin, INPUT);
  pinMode(button2Pin, INPUT);

  pinMode(ledPin, OUTPUT);
}
```

## PROGRAM

- WITHIN OUR loop STRUCTURE, WE CAN SET UP TWO MORE VARIABLES, AND SINCE THEY DON'T HAVE ASSIGNED VALUES THEY CAN LISTED ON A SINGLE LINE SEPARATED BY A COMMA.
- THEN WE SET THE VALUES OF THOSE VARIABLES TO THE DIGITAL STATE OF THE BUTTON PINS USING THE digitalRead FUNCTION.
- WE'LL SET UP ANOTHER if…else STATEMENT WHERE IF THE FIRST TEST IS TRUE THE LED WILL LIGHT UP, OTHERWISE IT WILL STAY OFF.
- WE CAN USE BOOLEAN LOGIC OPERATORS AS PART OF OUR TEST. THE BASIC BOOLEAN OPERATORS IN THE ARDUINO LANGUAGE ARE:
    - A == B MEANS "EQUIVALENT". THIS STATEMENT IS TRUE IF BOTH SIDES ARE THE SAME.
    - A && B MEANS "AND". THIS STATEMENT IS TRUE IF BOTH SIDES ARE TRUE.
    - A || B MEANS "OR". THIS STATEMENT IS TRUE IF EITHER SIDE IS TRUE.
    - !A MEANS "NOT". THIS STATEMENT IS TRUE IF A IS NOT TRUE.
- OUR BOOLEAN OPERATORS IN OUR CODE STATE THAT if BUTTON 1 OR BUTTON 2 ARE LOW, AND BUTTON 1 AND BUTTON 2 ARE NOT LOW, THEN WRITE HIGH TO THE LED PIN.
    - else WRITE LOW TO THE LED PIN.

```
void loop()
{
  int button1State, button2State;

  button1State = digitalRead(button1Pin);
  button2State = digitalRead(button2Pin);

  if (((button1State == LOW) || (button2State == LOW))
      && !
      ((button1State == LOW) && (button2State == LOW)))

  {
    digitalWrite(ledPin, HIGH);  // turn the LED on
  }
  else
  {
    digitalWrite(ledPin, LOW);  // turn the LED off
  }

}
```

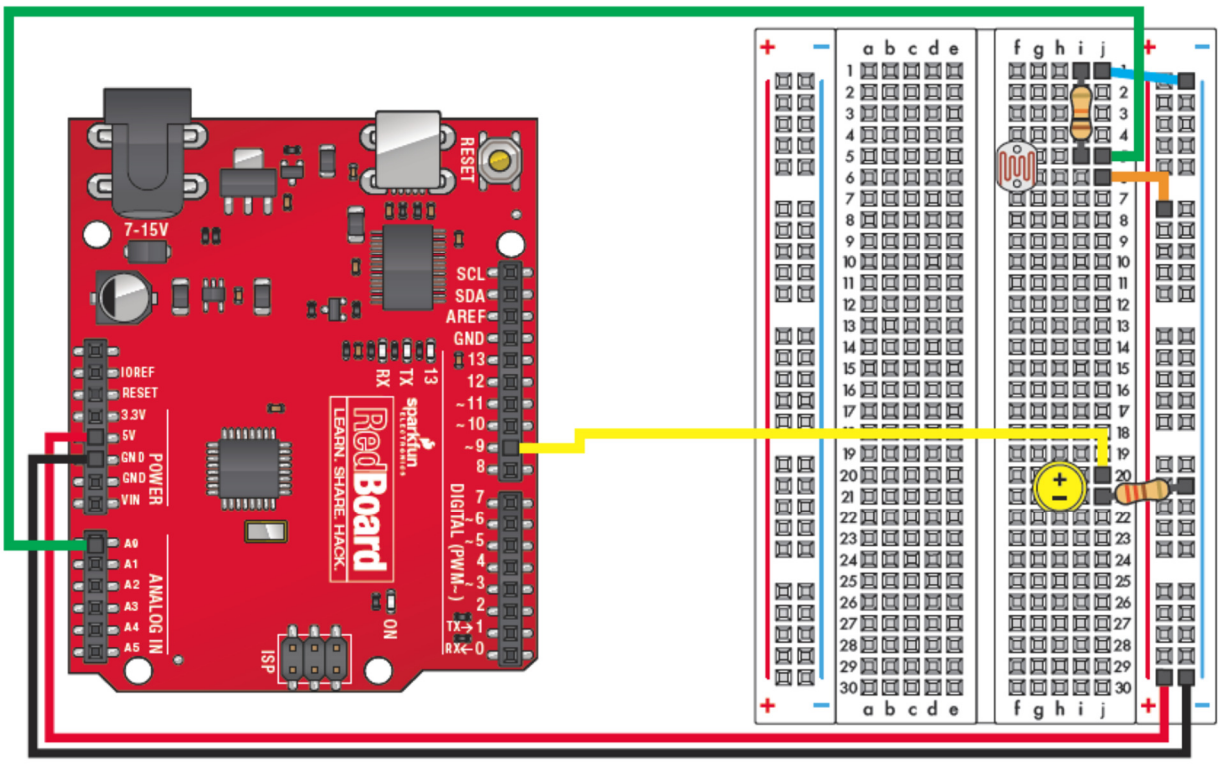
```
const int button1Pin = 2;
const int button2Pin = 3;
const int ledPin =  13;


void setup()
{
  pinMode(button1Pin, INPUT);
  pinMode(button2Pin, INPUT);

  pinMode(ledPin, OUTPUT);
}


void loop()
{
  int button1State, button2State;

  button1State = digitalRead(button1Pin);
  button2State = digitalRead(button2Pin);

  if (((button1State == LOW) || (button2State == LOW))
      && !
      ((button1State == LOW) && (button2State == LOW)))

  {
    digitalWrite(ledPin, HIGH);  // turn the LED on
  }
  else
  {
    digitalWrite(ledPin, LOW);  // turn the LED off
  }

}
```

# *Experiment 6*

In this experiment we will introduce an analog sensor, specifically a photoresistor which reacts to the amount of light that strikes its surface. A photoresistor changes its resistance based on the amount of light striking it, and as the resistance changes we can read the changes in voltage and program our board to react accordingly. Because sensors can take many values, we often have to convert them into values that our board can interpret and constrain them from negatively affecting the circuitry.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE THE PHOTORESISTOR AND AN LED ON THE BREADBOARD. CONNECT THE NEGATIVE SIDE OF THE LED TO GROUND USING A 330Ω RESISTOR.
- CONNECT ONE TERMINAL OF THE PHOTORESISTOR TO POSITIVE AND THE OTHER TO NEGATIVE USING A 1KΩ PULL-UP RESISTOR.
- CONNECT ANALOG PIN 0 TO THE NEGATIVE TERMINAL OF THE PHOTORESISTOR, AND DIGITAL PIN 9 TO THE POSITIVE SIDE OF THE LED.
    - NOTE THAT DIGITAL PIN 9 IS A PWM PIN.



## PROGRAM

- WE WILL SET UP GLOBAL VARIABLES FOR THE PINS WE WILL BE USING AS INTEGER CONSTANTS, AND A THIRD VARIABLE AS AN INTEGER WITHOUT A DECLARED VALUE.
- UNDER setup WE CAN SET UP OUR DIGITAL LED PIN AS AN OUTPUT USING pinMode.
    - AGAIN, WE DO NOT NEED TO DECLARE OUR ANALOG PIN SINCE IT CAN ONLY BE AN INPUT.

```
const int sensorPin = 0;
const int ledPin = 9;

int lightLevel

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

## PROGRAM

- WITHIN OUR loop STRUCTURE, WE SET OUR lightLevel VARIABLE EQUAL TO THE VOLTAGE READ FROM THE SENSOR PIN USING analogRead.
    - REMEMBER, THIS WILL RETURN A VALUE BETWEEN 0 (0V) AND 1023 (5V).
- WE WANT THE BRIGHTNESS OF THE LED TO CORRESPOND TO THE BRIGHTNESS OF THE LIGHT ON THE PHOTORESISTOR, BUT CAN ONLY WRITE VALUES BETWEEN 0 (0V AND 255 (5V) USING OUR DIGITAL PWM PIN.
- WE CAN map THE RANGE OF VALUES READ FROM THE ANALOG PIN TO THE RANGE OF VALUES WRITTEN TO THE DIGITAL PWM PIN.
    - THE map FUNCTION HAS 5 ARGUMENTS IN PARENTHESES: THE VARIABLE TO BE MAPPED, THE MININUM AND MAXIMUM VALUES OF THE INPUT RANGE, AND THE MINIMUM AND MAXIMUM VALUES OF THE OUTPUT RANGE.
        - map(variable, min_input, max_input, min_output, max_output)
- WE ALSO WANT TO constrain THE VALUES BEING WRITTEN TO THEIR MAXIMUM AND MINIMUM VALUES, SO THAT ANY STRANGE READINGS WON'T DAMAGE OUR BOARD.
- FINALLY WE CAN analogWrite OUR MAPPED VALUE TO THE ledPin.

```
void loop()
{
  lightLevel = analogRead(sensorPin);

  lightLevel = map(lightLevel, 0, 1023, 0, 255);
  lightLevel = constrain(lightLevel, 0, 255);

  analogWrite(ledPin, lightLevel);
}
```

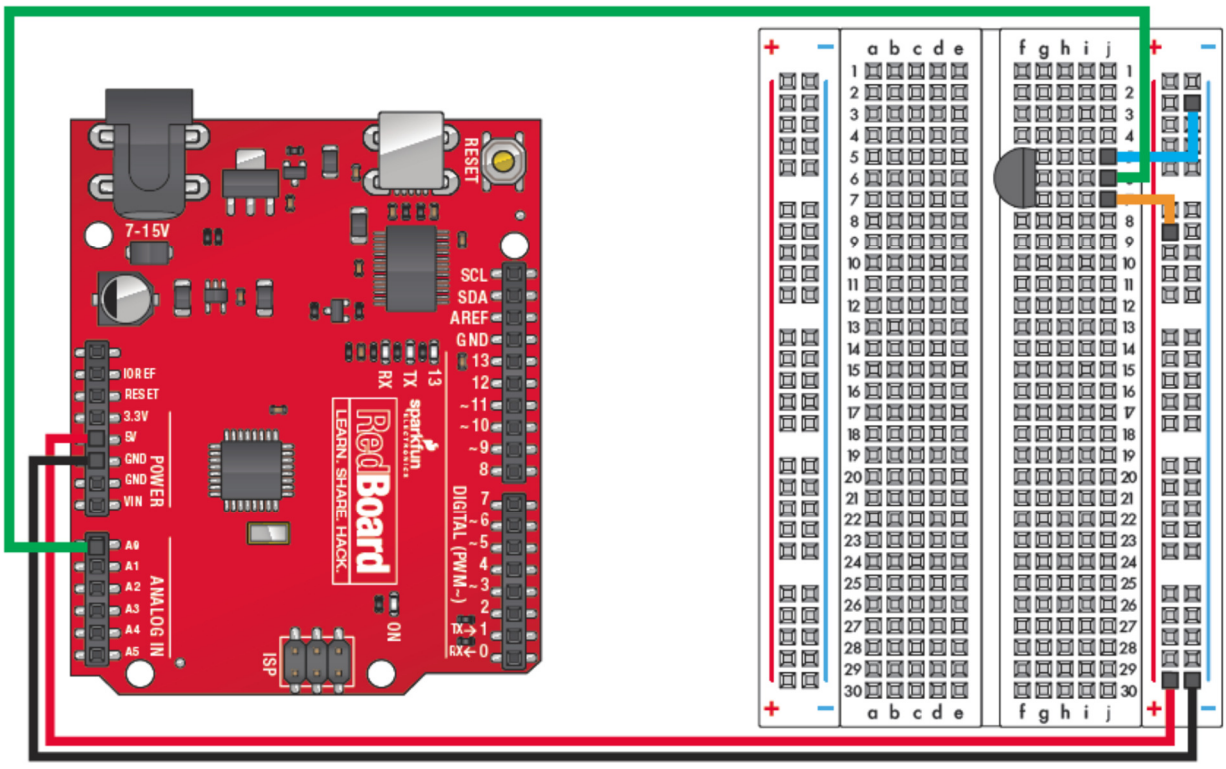
```
const int sensorPin = 0;
const int ledPin = 9;

int lightLevel, high = 0, low = 1023;

void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
  lightLevel = analogRead(sensorPin);

  lightLevel = map(lightLevel, 0, 1023, 0, 255);
  lightLevel = constrain(lightLevel, 0, 255);

  analogWrite(ledPin, lightLevel);
}
```

# *Experiment 7*

Using an analog temperature sensor, we can read data directly from the board using the same USB connection that we use to program the board. In order to have the measurements relayed to the computer, we will need to monitor the serial connection between the two and give it instructions on what data we want sent to the computer. To open a window to read the information , click the “Serial Monitor” button on the program window.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE THE TEMPERATURE SENSOR ON THE BREADBOARD.
  - NOTE THAT THE TEMPERATURE SENSOR LOOKS VERY MUCH LIKE A TRANSISTOR. THE TEMPERATURE SENSOR WILL HAVE “TMP” WRITTEN ON IT IN VERY SMALL LETTERS.
- CONNECT ONE TERMINAL OF THE TEMP. SENSOR TO POSITIVE AND THE OTHER TO NEGATIVE AS SHOWN IN THE DIAGRAM BELOW.
- CONNECT ANALOG PIN 0 TO THE MIDDLE TERMINAL OF THE TEMP. SENSOR.



## PROGRAM

- WE WILL SET UP GLOBAL VARIABLES FOR THE ANALOG PIN AS AN INTEGER CONSTANTS.
- UNDER setup WE NEED TO OPEN A Serial CONNECTION TO THE BOARD AND SPECIFY THE "BAUD RATE" AT WHICH IT SHOULD RUN. THIS ALLOWS THE BOARD TO SEND DATA BACK TO THE COMPUTER WHILE THE SKETCH IS RUNNING.
    - WE USE THE Serial.begin COMMAND, AND THE BAUD RATE WILL ALMOST ALWAYS BE 9600.

```
const int temperaturePin = 0;


void setup()
{
    Serial.begin(9600);
}
```

## PROGRAM

- INSIDE THE loop, WE CREATE “FLOATING” VARIABLES (float) FOR THE VOLTAGE, CELSIUS TEMP. AND FAHRENHEIT TEMP
    - FLOATING VARIABLES CAN HAVE DECIMAL POINTS, UNLIKE INTEGER VARIABLES.
- WE SET OUR VOLTAGE VARIABLE TO THE analogRead VALUE FROM THE SENSOR MULTIPLIED BY A CONSTANT SPECIFIC TO THE SENSOR. THEN WE SET OUR TEMPERATURE READINGS BASED ON THE VOLTAGE VALUE.
- THE Serial.print COMMAND TELLS THE BOARD WHAT TO DISPLAY IN THE SERIAL MONITOR WINDOW.
    - FIRST IT WILL PRINT THE TEXT IN QUOTES (voltage: ) FOLLOWED BY THE VALUE OF THE voltage VARIABLE. THEN THE TEXT FOR degC: AND degF: EACH FOLLOWED BY THEIR VARIABLE VALUES.
- THE FINAL Serial.println COMMAND TELLS THE MONITOR THAT IT IS THE END OF THE LINE AND IT SHOULD START A NEW LINE.

```
void loop()
{

  float voltage, degreesC, degreesF;

  voltage = analogRead(temperaturePin)* 0.004882814;

  degreesC = (voltage - 0.5) * 100.0;

  degreesF = degreesC * (9.0/5.0) + 32.0;

  Serial.print("voltage: ");
  Serial.print(voltage);
  Serial.print("  deg C: ");
  Serial.print(degreesC);
  Serial.print("  deg F: ");
  Serial.println(degreesF);

  delay(1000);
}
```

```
const int temperaturePin = 0;


void setup()
{
    Serial.begin(9600);
}

void loop()
{

  float voltage, degreesC, degreesF;

  voltage = analogRead(temperaturePin)* 0.004882814;

  degreesC = (voltage - 0.5) * 100.0;

  degreesF = degreesC * (9.0/5.0) + 32.0;

  Serial.print("voltage: ");
  Serial.print(voltage);
  Serial.print("  deg C: ");
  Serial.print(degreesC);
  Serial.print("  deg F: ");
  Serial.println(degreesF);

  delay(1000);
}
```

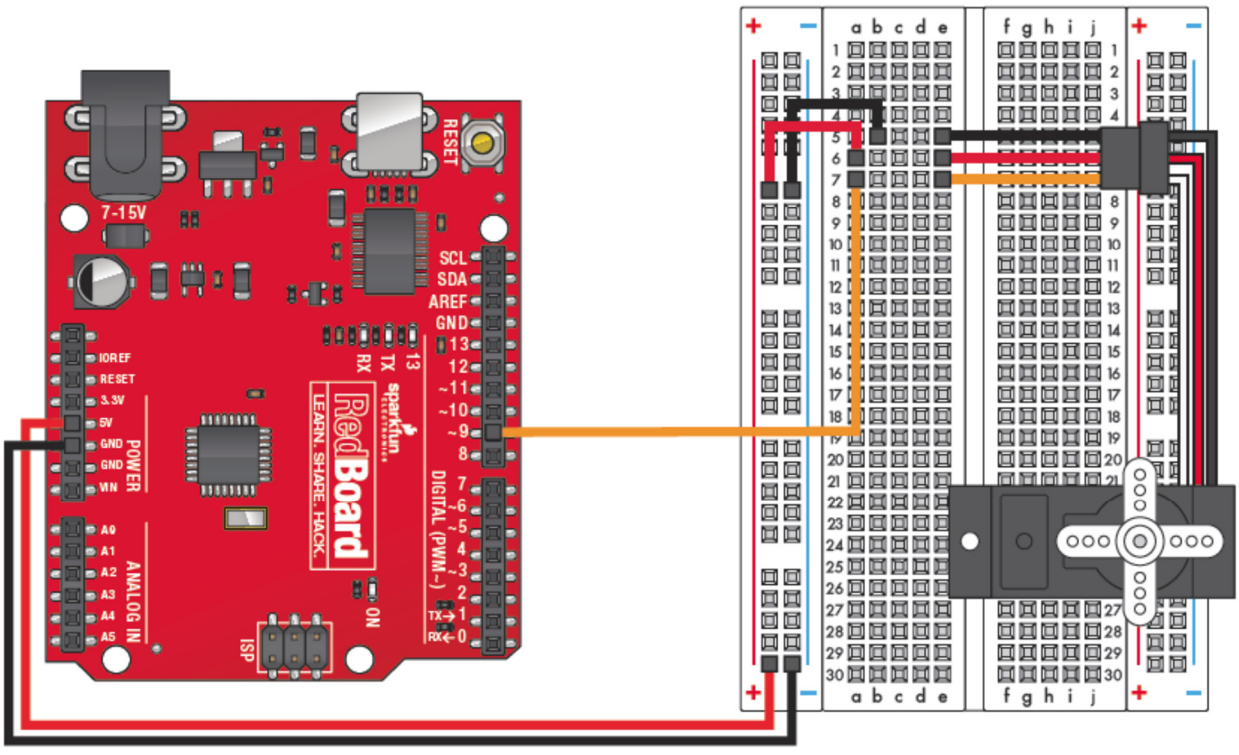
# *Experiment 8*

Servo motors are motors which can be programmed to rotate by a specified angle and stop at that point. They are often used in robotics applications to control movable arms and very precisely geared mechanisms. However, these motors can be very difficult to program from scratch, so in this experiment we will introduce the concept of using "libraries" to greatly simplify the programming aspects of difficult tasks.

Our servo motors can move to any specified angle between 0° and 180°, and we can control their speed using Pulse Width Modulation.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- FROM THE SERVO MOTOR, CONNECT THE RED WIRE TO POSITIVE AND THE BLACK WIRE TO GROUND
- THE WHITE WIRE FROM THE SERVO MOTOR SHOULD BE CONNECTED TO DIGITAL PIN 9 ON THE BOARD.
    - NOTE THAT DIGITAL PIN 9 IS A PWM PIN.



## PROGRAM

- BEFORE ANYTHING ELSE, WE NEED TO #include THE SERVO LIBRARY INTO OUR PROGRAM. THE LIBRARY WE'LL USE IS CALLED Servo.h.
- THIS LIBRARY ALSO REQUIRES US TO NAME OUR Servo MOTOR, WHICH WE'LL SIMPLY CALL servo1.
- UNDER setup WE NEED TO SPECIFY WHICH PIN OUR SERVO IS CONNECTED TO, THE LIBRARY REQUIRES THE attach COMMAND AS FOLLOWS:
    - servo_name.attach(pin)

```
#include <Servo.h>  // servo library

Servo servo1;  // servo control object

void setup()
{
  servo1.attach(9);
}
```

## PROGRAM

- NOW THAT THE LIBRARY IS INCLUDED, IT GREATLY SIMPLIFIES THE PROGRAMMING INSIDE OUR loop.
- WE CON SIMPLY WRITE AN ANGLE (IN DEGREES) TO OUR SERVO AND THE MOTOR WILL MOVE TO THAT ANGLE:
    - servo_name.write(angle)
- BECAUSE THE MOTOR TAKES TIME TO MOVE TO THE SPECIFIED ANGLE, ITS GOOD PRACTICE TO INCLUDE A delay BETWEEN INSTRUCTIONS. THIS GIVES THE MOTOR TIME TO MOVE BEFORE THE NEXT INSTRUCTION IS GIVEN.
- WE'LL MAKE OUR SERVO MOTOR TRAVEL FROM 0° TO 90°, THEN FROM 90° TO 180°, AND FINALLY BACK TO 0°.
    - NOTE THAT THE MOTOR WILL BE TRAVELLING AT FULL SPEED, SINCE WE HAVE GIVEN NO INSTRUCTIONS TO SLOW IT DOWN.

```
void loop()
{
  servo1.write(90);

  delay(1000);

  servo1.write(180);

  delay(1000);

  servo1.write(0);

  delay(1000);
}
```

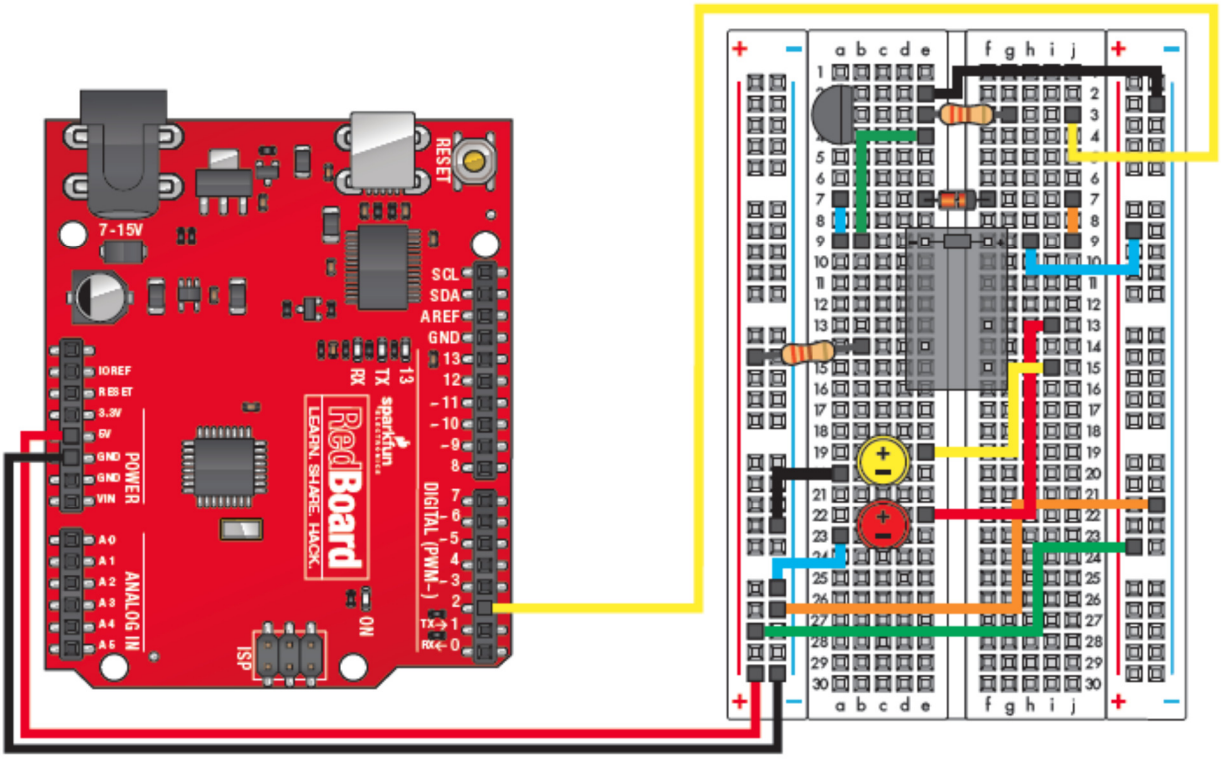
```
#include <Servo.h>  // servo library

Servo servo1;  // servo control object

void setup()
{
  servo1.attach(9);
}


void loop()
{
  int position;

  servo1.write(90);

  delay(1000);

  servo1.write(180);

  delay(1000);

  servo1.write(0);

  delay(1000);
}
```

# *Experiment 9*

Most boards are limited in the amount of current they can supply to a device, so some of the more power-hungry devices will need an external power supply in order to run. However we can still use our board in conjunction with transistors and relays to control these devices. Transistors can switch current very quickly, relays are a bit slower but can handle more current, both can be controlled by an Arduino or similar board.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE THE RELAY ON THE BREADBOARD SO THAT IT STRADDLES THE CENTER GAP.
- WE'LL USE THE TRANSISTOR TO CONTROL THE RELAY, SO CONNECT THE TRANSISTOR AS SHOWN IN THE DIAGRAM WITH THE MIDDLE TERMINAL (BASE) CONNECTED TO DIGITAL PIN 2 OF THE BOARD.
- CONNECT THE TWO LEDS TO THE OUTPUT PINS OF THE RELAY, AND CONNECT THE INPUT PIN TO POWER USING A 330 RESISTOR.
- WHEN SWITCHING THE RELAY WE WILL USE A DIODE TO PROTECT OUR BOARD FROM "FLY BACK" CURRENT.



## PROGRAM

- WE WILL #include THE SERVO LIBRARY CALLED Servo.h INTO OUR PROGRAM, THEN NAME OUR Servo MOTOR servo1.
- WE'LL CREATE TWO const int FOR THE relayPin AND A timeDelay VARIABLE.
- UNDER setup WE WILL USE pinMode TO SET OUR relayPin AS AN OUTPUT.

```
const int relayPin = 2;
const int timeDelay = 1000;

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

## PROGRAM

- SINCE WE WILL JUST BE SWITCHING THE RELAY ON AND OFF THROUGH THE TRANSISTOR, OUR loop WILL SIMPLY CONSIST OF HIGH AND LOW COMMANDS.
    - WE'LL USE digitalWrite TO SEND THE relayPin HIGH.
    - INCLUDE A BRIEF delay.
    - THEN USE digitalWrite TO SEND THE relayPin LOW.
    - INCLUDE A delay.

```
void loop()
{
  digitalWrite(relayPin, HIGH);

  delay(timeDelay);

  digitalWrite(relayPin, LOW);

  delay(timeDelay);
}
```

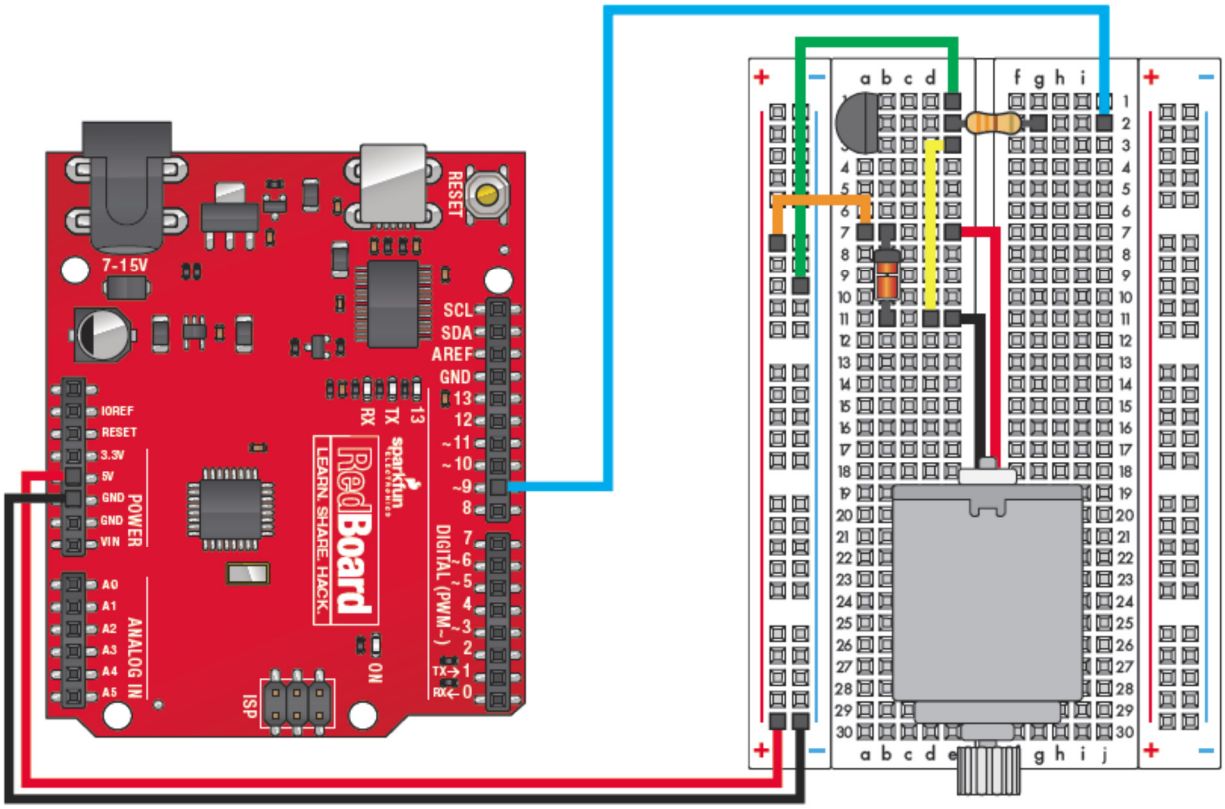
```
const int relayPin = 2;
const int timeDelay = 1000;

void setup()
{
  pinMode(relayPin, OUTPUT);
}


void loop()
{
  digitalWrite(relayPin, HIGH);

  delay(timeDelay);

  digitalWrite(relayPin, LOW);

  delay(timeDelay);
}
```

# *Experiment 10*

There are some devices for which our board will not be able to produce enough electrical current to run, such as motors and actuators. In these cases we can use a transistor as a switch for turning on current from a more powerful supply. Transistors switch fast enough that we can even use PWM control to adjust the speeds of the motors.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE THE TRANSISTOR ON THE BREADBOARD AS SHOWN IN THE DIAGRAM AND CONNECT DIGITAL PIN 9 TO THE CENTER TERMINAL THROUGH A 330Ω RESISTOR.
- CONNECT THE NEGATIVE (BLACK) WIRE TO THE TRANSISTOR AND THE POSITIVE (RED) WIRE TO POSITIVE POWER.
- PLACE A DIODE BETWEEN THE POSITIVE AND NEGATIVE MOTOR CONNECTIONS.
    - THE DIODE IS TO PREVENT “FLY-BACK” CURRENT, WHICH FLOWS IN THE OPPOSITE DIRECTION WHEN THE MOTOR TURNS OFF.



## PROGRAM

- DC MOTORS RUN WHEN AN APPROPRIATE VOTAGE IS APPLIED, SO WE DO NOT NEED ANY TYPE OF LIBRARY TO PROGRAM THEM.
- WE'LL CREATE A const int FOR THE PIN CONNECTED TO THE MOTOR, AND UNDER setup WE CAN SET THAT DIGITAL PIN TO BE AN OUTPUT.
- WE WILL ALSO OPEN THE Serial CONNECTION SO THAT WE CAN USE THE SERIAL MONITOR TO SET THE SPEED OF THE MOTOR USING PWM.

```
const int motorPin = 9;


void setup()
{
  pinMode(motorPin, OUTPUT);

  Serial.begin(9600);
}
```

## PROGRAM

- INSIDE OUR loop WE CREATE THE int VARIABLE speed AND WRITE INSTRUCTIONS TO THE SERIAL MONITOR USING THE Serial.println COMMAND.
    - NOTE THAT THE FINAL Serial.println COMMAND IS LEFT BLANK. THIS IS THE LINE THAT WILL READ THE NUMBER WE TYPE IN TO THE SERIAL MONITOR.
- WE WANT THIS PROGRAM TO KEEP RUNNING, BUT NOT KEEP WRITING THE SERIAL MONITOR INSTRUCTIONS OVER FOR EVERY LOOP, SO WE'LL CREATE AN "INFINITE" while LOOP FOR THE REST OF THE PROGRAM.
    - while(true) IS ALWAYS TRUE, SO THIS LOOP WILL NEVER END.
- NEXT OUR SKETCH ASKS IF THERE IS AN available NUMBER GREATER THAN ZERO IN THE BLANK LINE OF THE SERIAL MONITOR, AND while THERE IS WE WILL PULL THE INTEGER INTO OUR SKETCH USING THE parseInt() COMMAND.
- WE WANT TO constrain OUR speed IN CASE SOMETHING STRANGE IS ENTERED.
- THE MOTOR SPEED IS THEN DISPLAYED ON THE SERIAL MONITOR, AND WRITTEN TO THE MOTOR USING THE analogWrite COMMAND.

```
void loop()
{
  int speed;

  Serial.println("Type a speed (0-255) into the box above,");
  Serial.println("then click [send] or press [return]");
  Serial.println();

  while(true)
  {
    while (Serial.available() > 0)
    {
      speed = Serial.parseInt();

      speed = constrain(speed, 0, 255);

      Serial.print("Setting speed to ");
      Serial.println(speed);

      analogWrite(motorPin, speed);
    }
  }
}
```

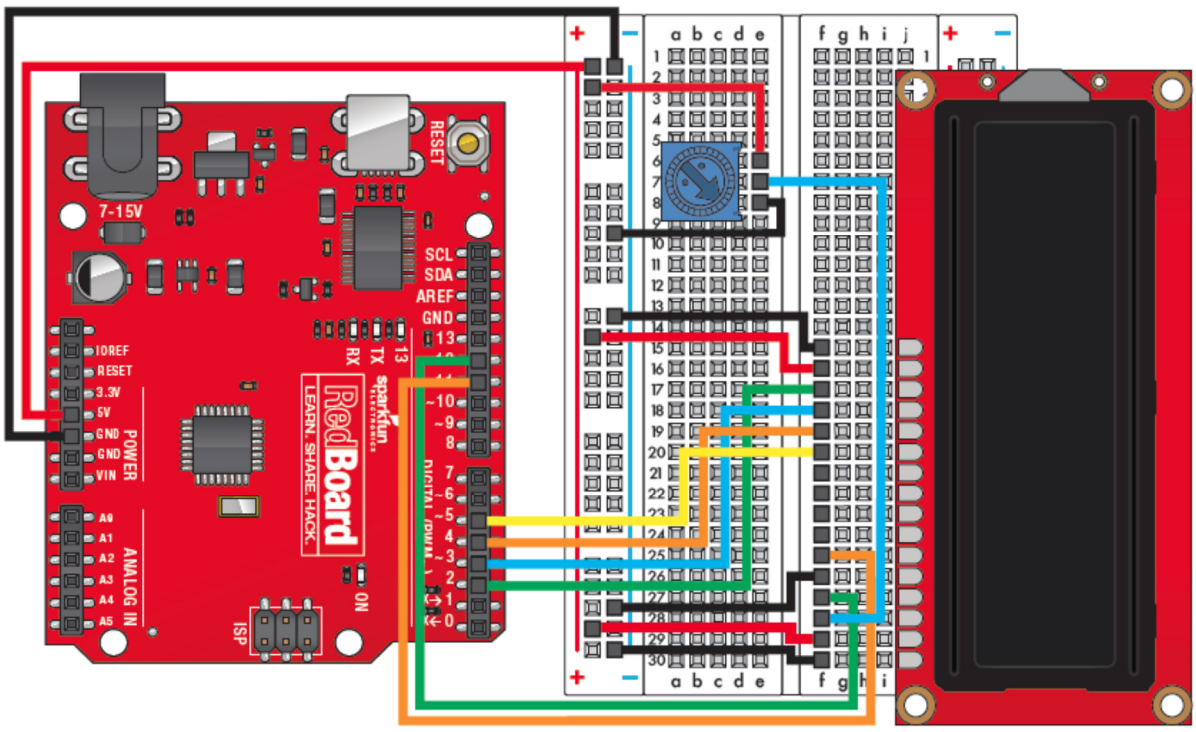
```ino
const int motorPin = 9;


void setup()
{
  pinMode(motorPin, OUTPUT);

  Serial.begin(9600);
}
void loop()
{
  int speed;

  Serial.println("Type a speed (0-255) into the box above,");
  Serial.println("then click [send] or press [return]");
  Serial.println();

  while(true)
  {
    while (Serial.available() > 0)
    {
      speed = Serial.parseInt();

      speed = constrain(speed, 0, 255);

      Serial.print("Setting speed to ");
      Serial.println(speed);

      analogWrite(motorPin, speed);
    }
  }
}
```

# *Experiment 11*

Our board can also be programmed to display information on an LCD screen. Since programming such devices can be difficult, we will again rely on libraries to help us. By using the LCD library we can display measurements and results on the LCD screen similar to the way we have displayed results on the serial monitor earlier.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE THE LCD SCREEN ON THE BREADBOARD AS SHOWN AND CONNECT THE POWER AND DIGITAL PINS TO ITS TERMINALS AS SHOWN IN THE DIAGRAM BELOW.
- PLACE THE POTENTIOMETER ON THE BREADBOARD, CONNECT THE OUTSIDE TERMINALS TO POSITIVE AND GROUND AND CONNECT THE MIDDLE TERMINAL TO THE LCD SCREEN AS SHOWN IN THE DIAGRAM.



## PROGRAM

- FIRST WE NEED TO #include THE LCD LIBRARY, WHICH IS CALLED LiquidCrystal.h.
- FOLLOWING THE LIBRARIES INSTRUCTIONS, WE CAN DESIGNATE THE lcd PINS THAT WILL BE USED.
- WE CAN begin OUR setup BY SETTING THE SIZE OF OUR LCD DISPLAY. OUR DISPLAY WILL ALLOW 16 CHARACTERS PER LINE, WITH 2 LINES AVAILABLE.
- IN CASE OF ANY REMNANT DATA FROM ANOTHER SKETCH, WE CAN clear THE SCREEN.
- THEN WE CAN EXECUTE THE print COMMAND, WITH THE TEXT TO BE PRINTED IN QUOTES.
    - WITHOUR OTHER INSTRUCTION, OUR TEXT WILL PRINT AT THE BEGINNING ($0^{TH}$ CHARACTER) OF THE FIRST LINE ($0^{TH}$ LINE).
    - BECAUSE THIS IS ALL UNDER setup, IT WILL ONLY RUN ONCE AT THE BEGINNING OF THE SKETCH.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);

void setup() {
  lcd.begin(16, 2);
  lcd.clear();
  lcd.print("hello, world!");
}
```

## PROGRAM

- THE PART OF THE DISPLAY THAT WILL CONTINUE REPEATING MUST BE PLACED UNDER THE loop COMMAND.
- SINCE OUR FIRST LINE WAS COVERED UNDER SETUP (POSITON 0,0), WE'LL WANT TO SET OUR CURSOR POSITION TO THE BEGINNING OF THE SECOND LINE.
    - WE USE THE setCursor COMMAND TO START AT THE 0 POSITION (1ST CHARACTER) OF LINE 1 (2ND LINE).
- OUR SECOND print COMMAND WILL ASK THE BOARD TO SHOW THE NUMBER OF SECONDS THE PROGRAM HAS BEEN RUNNING.
    - TIME IS MEASURED IN MILLISECONDS BY THE BOARD, WE CAN REQUEST THAT TIME USING THE millis() COMMAND.

```
void loop() {

  lcd.setCursor(0,1);

  lcd.print(millis()/1000);

}
```

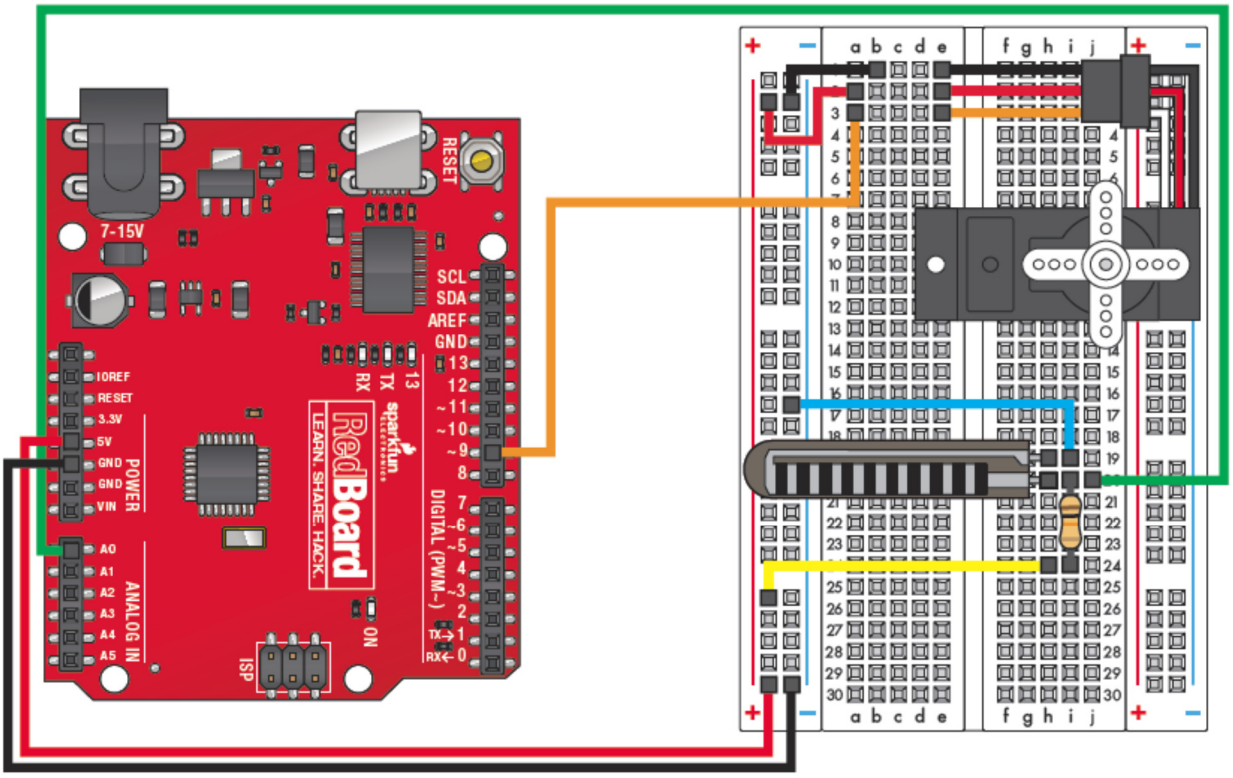
```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);

void setup() {
  lcd.begin(16, 2);
  lcd.clear();
  lcd.print("hello, world!");
}

void loop() {

  lcd.setCursor(0,1);

  lcd.print(millis()/1000);

}
```

# *Experiment 12*

We can combine analog inputs with digital outputs to develop a system that is reactive to outside stimulus. Flex sensors are devices which change their resistance as they are bent, so it would make sense to combine one with a servo motor which might be used in a robotics setting. The difficulty with this is often calibrating the sensor's readings with the desired output to the servo motor, which will be addressed in this experiment.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- FROM THE SERVO MOTOR, CONNECT THE RED WIRE TO POSITIVE, BLACK WIRE TO GROUND, AND WHITE WIRE TO DIGITAL PIN 9 ON THE BOARD.
- CONNECT THE FLEX SENSOR TO THE BREADBOARD WITH ONE TERMINAL CONNECTED TO GROUND AND THE OTHER CONNECTED TO POSITIVE THROUGH A 10K PULL-UP RESISTOR.
- FINALLY CONNECT THE POSITIVE TERMINAL OF THE FLEX SENSOR TO ANALOG PIN 0.



## PROGRAM

- WE WILL #include THE SERVO LIBRARY CALLED Servo.h INTO OUR PROGRAM, THEN NAME OUR Servo MOTOR servo1.
- WE'LL ALSO CREATE A const int FOR THE PIN CONNECTED TO THE FLEX SENSOR.
- UNDER setup WE WILL NEED TO attach OUR SERVO TO THE CORRECT DIGITAL PIN.
- WE WILL ALSO OPEN THE Serial CONNECTION SO THAT WE CAN READ THE VALUES BEING USED BY THE BOARD.

```
#include <Servo.h>
Servo servo1;
const int flexpin = 0;

void setup()
{
  Serial.begin(9600);

  servo1.attach(9);
}
```

## PROGRAM

- INSIDE OUR loop WE CREATE TWO MORE int VARIABLES TO DESCRIBE THE POSITIONS OF OUR FLEX SENSOR AND SERVO MOTOR.
- THE flexposition VARIABLE IS DEFINED AS THE analogRead VALUE OF THE flexpin.
    - THIS VALUE WILL BE BETWEEN 0 (0V) AND 1023 (5V).
- OUR servoposition VARIABLE WILL BE THE ANGLE IT MOVES TO, WHICH MUST BE BETWEEN 0° AND 180°. THUS WE'LL NEED TO map OUR flexposition VALUES TO THE RANGE OF POSSIBLE ANGLES.
    - INITIALLY WE DON'T KNOW WHAT OUR FLEXPOSITION VALUES WILL BE, SO WE SIMPLY GUESS AT A RANGE FROM 600 TO 900.
- WE ALSO NEED TO constrain OUR servoposition BETWEEN 0 AND 180.
- THEN WE CAN write THE servopositionTO THE SERVO AS AN ANGLE.
- WE'LL TELL OUR Serial MONITOR TO print THE VALUES OF THE FLEXPOSITION AND SERVOPOSITION VALUES SO THAT WE CAN MAP THEM MORE PRECISELY.
    - ONCE WE SEE THE ACTUAL RANGE OF VALUES THE FLEX SENSOR GIVES US WE CAN BETTER CALIBRATE IT TO THE SERVO MOTOR.
- FINALLY WE'LL ADD A BRIEF delay TO GIVE THE MOTOR TIME TO CATCH UP.

```
void loop()
{
  int flexposition;
  int servoposition;

  flexposition = analogRead(flexpin);

  servoposition = map(flexposition, 600, 900, 0, 180);
  servoposition = constrain(servoposition, 0, 180);

  servo1.write(servoposition);

  Serial.print("sensor: ");
  Serial.print(flexposition);
  Serial.print("  servo: ");
  Serial.println(servoposition);

  delay(20);
}
```

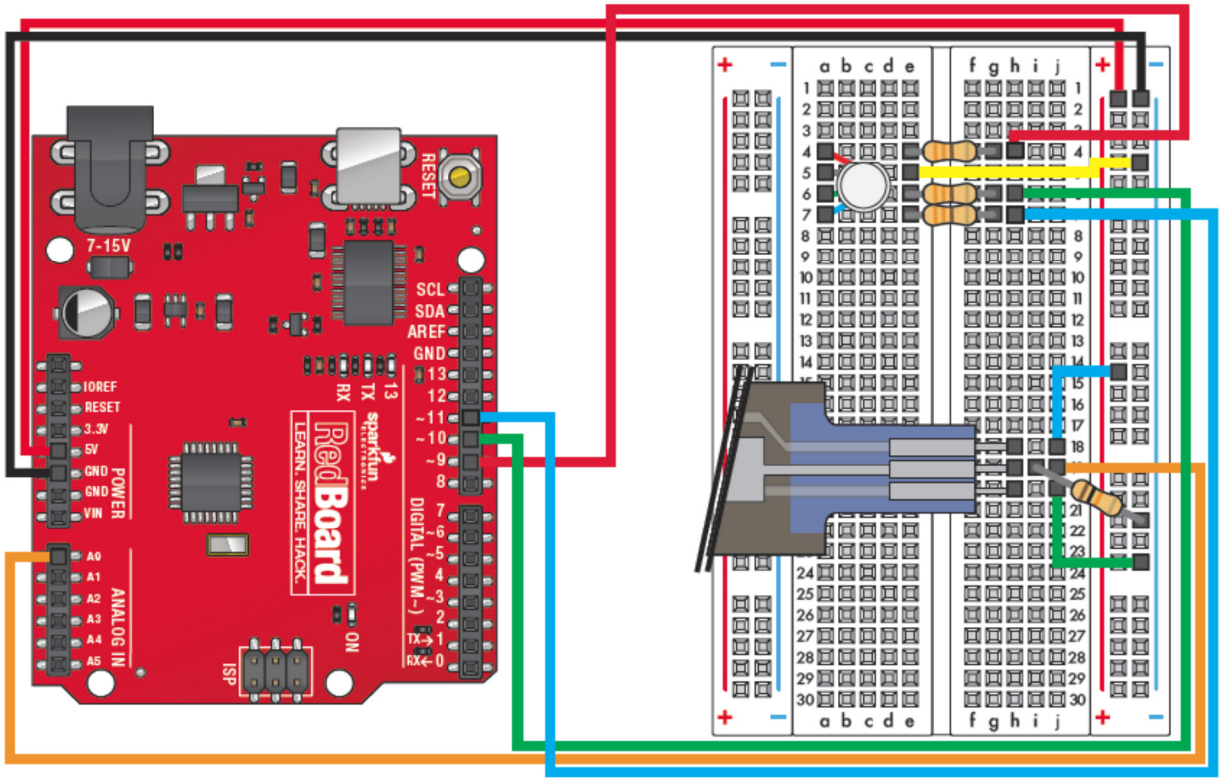
```ino
#include <Servo.h>
Servo servo1;
const int flexpin = 0;

void setup()
{
  Serial.begin(9600);

  servo1.attach(9);
}

void loop()
{
  int flexposition;
  int servoposition;

  flexposition = analogRead(flexpin);

  servoposition = map(flexposition, 600, 900, 0, 180);
  servoposition = constrain(servoposition, 0, 180);

  servo1.write(servoposition);

  Serial.print("sensor: ");
  Serial.print(flexposition);
  Serial.print("  servo: ");
  Serial.println(servoposition);

  delay(20);
}
```

# *Experiment 13*

Another type of analog sensor that we can use is called a “soft potentiometer”. A soft potentiometer (or “soft pot”) changes its resistance depending on where it is touched, much like the screen of a smart phone. We can read the variation in voltage that occurs when the soft pot is touched in different positions and map it to a digital output to control another device, in this case we will use PWM output to control the colors of an RGB LED.

## CIRCUIT SETUP

- CONNECT THE 5V POWER (+) AND THE GROUND (-) PINS ON THE REDBOARD TO THE POSITIVE AND NEGATIVE BREADBOARD RAILS.
- PLACE THE RGB LED ON THE BREADBOARD AND CONNECT THE NEGATIVE PIN (LONGEST PIN) TO GROUND.
- CONNECT THE OTHER THREE PINS OF THE RGB LED TO DIGITAL PINS 9, 10, AND 11 ON THE BOARD THROUGH 330Ω RESISTORS.
- CONNECT THE SOFT POTENTIOMETER TO THE BREADBOARD WITH THE OUTSIDE TERMINALS CONNECTED TO GROUND AND POSITIVE.
- CONNECT THE MIDDLE TERMINAL TO GROUND THROUGH A 10KΩ RESISTOR, AND ALSO CONNECT TO ANALOG PIN 0 ON THE BOARD.



## PROGRAM

- WE CAN CREATE A const int FOR EACH OF THE COLOR PINS FOR THE LED, AND ALSO FOR THE ANALOG SENSOR PIN CONNECTED TO THE SOFT POT.
- WE CAN ALSO CREATE int VARIABLES FOR EACH OF THE COLORS TO BE WRITTEN TO THE RGB LED.
- UNDER setup WE WILL SET OUR DIGITAL PINS AS OUTPUTS USING pinMode.

```
const int RED_LED_PIN = 9;    // Red LED Pin
const int GREEN_LED_PIN = 10; // Green LED Pin
const int BLUE_LED_PIN = 11;  // Blue LED Pin

const int SENSOR_PIN = 0;      // Analog input pin

int redValue, greenValue, blueValue;


void setup()
 {
 pinMode(RED_LED_PIN, OUTPUT)
 pinMode(GREEN_LED_PIN, OUTPUT)
 pinMode(BLUE_LED_PIN, OUTPUT)
 }
```

## PROGRAM

- INSIDE OUR loop WE CREATE AN int VARIABLE FOR RGBPOSITION AND DEFINE IT AS THE analogRead VALUE OF THE sensorpin.
    - THIS VALUE WILL BE BETWEEN 0 (0V) AND 1023 (5V).
- WE WILL BREAK THE VALUES FROM THE SOFT POT INTO THREE RANGES (ONE FOR EACH COLOR), THEN map THOSE RANGES ONTO VALUES TO SEND TO THE LED.
- FIRST WE CREATE A RED PEAK CENTERED AT 0 (RANGE FROM 682 TO 1023 AND 0 TO 341) BY mapPING THOSE RGBposition VALUES AND constrainING THEM TO VALUES FROM 0 TO 255)
    - BY ADDING THESE TWO VALUES TOGETHER WE GET OUR RED PEAK AT 0.
- WE CAN SIMPLIFY THIS PROCEDURE BY INCLUDING OUR map AND constrain FUNCTIONS ON A SINGLE LINE TO GET OUR GREEN PEAK (AT 341) AND OUR BLUE PEAK (AT 682).
- THEN WE USE analogWrite TO SEND EACH OF OUR VALUES TO THE CORRESPONDING COLOR PINS FOR THE RGB LED.

```
void loop()
{
  int RGBposition
  RGBposition= analogRead(0);

  int mapRGB1, mapRGB2, constrained1, constrained2;

  mapRGB1 = map(RGBposition, 0, 341, 255, 0);
  constrained1 = constrain(mapRGB1, 0, 255);

  mapRGB2 = map(RGBposition, 682, 1023, 0, 255);
  constrained2 = constrain(mapRGB2, 0, 255);

  redValue = constrained1 + constrained2;

  greenValue = constrain(map(RGBposition, 0, 341, 0, 255), 0, 255)
             - constrain(map(RGBposition, 341, 682, 0,255), 0, 255);

  blueValue = constrain(map(RGBposition, 341, 682, 0, 255), 0, 255)
            - constrain(map(RGBposition, 682, 1023, 0, 255), 0, 255);

  analogWrite(RED_LED_PIN, redValue);
  analogWrite(GREEN_LED_PIN, greenValue);
  analogWrite(BLUE_LED_PIN, blueValue);
}
```

```
const int RED_LED_PIN = 9;    // Red LED Pin
const int GREEN_LED_PIN = 10; // Green LED Pin
const int BLUE_LED_PIN = 11;  // Blue LED Pin

const int SENSOR_PIN = 0;      // Analog input pin

int redValue, greenValue, blueValue;


void setup()
 {
 pinMode(RED_LED_PIN, OUTPUT)
 pinMode(GREEN_LED_PIN, OUTPUT)
 pinMode(BLUE_LED_PIN, OUTPUT)
 }

void loop()
{
  int RGBposition
  RGBposition= analogRead(0);

  int mapRGB1, mapRGB2, constrained1, constrained2;

  mapRGB1 = map(RGBposition, 0, 341, 255, 0);
  constrained1 = constrain(mapRGB1, 0, 255);

  mapRGB2 = map(RGBposition, 682, 1023, 0, 255);
  constrained2 = constrain(mapRGB2, 0, 255);

  redValue = constrained1 + constrained2;

  greenValue = constrain(map(RGBposition, 0, 341, 0, 255), 0, 255)
             - constrain(map(RGBposition, 341, 682, 0,255), 0, 255);

  blueValue = constrain(map(RGBposition, 341, 682, 0, 255), 0, 255)
            - constrain(map(RGBposition, 682, 1023, 0, 255), 0, 255);

  analogWrite(RED_LED_PIN, redValue);
  analogWrite(GREEN_LED_PIN, greenValue);
  analogWrite(BLUE_LED_PIN, blueValue);
}
```