Constructing and Controlling Circuits

Authors: Dorottya B. Noble[1,2], Alejandro Virrueta[1,3], Joshua Young[9], Douglas B. Noble[4], Corey S. O'Hern[1,3,5,6,7], and Lynne Regan[1,2,7,8]

Affiliations: [1]Integrated Graduate Program in Physical and Engineering Biology, [2]Department of Molecular Biophysics and Biochemistry, [3]Department of Mechanical Engineering and Materials Science, [4]Office of Environmental Health and Safety, [5]Department of Physics, [6]Department of Applied Physics, [7]Graduate Program in Computational Biology and Bioinformatics, [8]Department of Chemistry, Yale University, New Haven, CT 06520, USA; [9]Hopkins School, New Haven, CT 06515, USA

Keywords: circuits, microcontroller, Arduino, hands-on, programming

Abstract: We describe simple hands-on investigations to introduce middle school students (ages 12-14) to circuits and their components, and how to control circuits using software. We discuss how commercially available kits can be used to build circuits and understand different circuit components. The module also includes an introduction to Arduino programming, culminating in the students implementing and playing a computer-controlled game using a circuit.

1) Introduction and Learning Objectives

This module introduces students to electronics by familiarizing them with electrical circuits. In "Part 1. Circuit basics," students will learn the basics about circuits, will learn the function of circuit components, will be able to use these components to build a circuit, and will understand the difference between analog and digital signals. In "Part 2. Microcontrollers: Combining hardware and software," students will use a microcontroller to read in sensor values and manipulate them using software and will understand how to interface circuits with microcontrollers. In "Part 3. Software-based circuit game," the first two parts are combined to build an LED-based game. The students will learn how simple circuits can be combined to generate more complex ones, will recognize and understand simple segments of computer code, and will understand how the code controls an LED game.

2) Preparations prior to using the module

Investigate the Thinker Kit, builds the circuits to be used in this module and gains some familiarity with the open source Arduino software, which must be installed on the computer(s) to be used (https://www.arduino.cc/en/Main/Software).

Explore the *Experiment Guide for SparkFun Thinker Kit* website (https://learn.sparkfun.com/tutorials/experiment-guide-for-the-sparkfun-tinker-kit) for more detailed descriptions of the components of the kit, links and instructions to download and install Arduino.

Based on our pilot testing, we recommend that students always use resistors in series with other electrical components, and that extra LEDs are purchased, because students often build circuits incorrectly in the beginning and burn out the LEDs (Table S1).

3) Lesson Plans

Part 1. Circuit basics (~ 1-1.5h). Students are introduced to 1) the concept of a circuit, 2) the different components of a circuit, and 3) the Breadboard and microcontroller, and how to power circuits using them.

What is a circuit: Explain that a circuit is a continuous path around which electricity, or current, flows. Circuit components can perform useful functions, such as illuminating a light bulb, switching a light bulb on and off, or dimming a light bulb.

Components of a circuit: Explain the function of each component, encourage the students to examine them, and show how each component is represented schematically (Figure 1).

Basics of the breadboard and microcontroller: Students must understand the layout of the breadboard (Figure 2) and microcontroller (Figure 3). Explain the organization of the breadboard and show important pins in the microcontroller. The breadboard is organized into columns a-j with a "+/power" and a "-/ground" column adjacent to columns a and j.

The computer provides power to the circuit, via the microcontroller, although in part 1, batteries (using the battery holder provided in the kit) are sufficient. For Part 1, point out the pins located along the "Power" portion of the microcontroller, labeled "5V" and "GND", and the USB and battery pack connectors (see Figure 3).

Building a circuit to power an LED: For the simple circuits in Part 1, provide students only with the circuit diagram, as opposed to the Breadboard view (Figures 1 and 4). *Note that there are multiple ways to configure a circuit using the Thinker Kit, and different groups may arrive at the same result using different configurations. We show one possibility in Figure 1 and subsequent figures.*

Students should first connect the "5V" pin on the microcontroller with the socket in row 1, column "+/power". Then connect the "GDN" pin on the microcontroller with the socket in row 1, column "-/ground". Then provide power to the section of the breadboard where the circuit components can be added. They can use a wire to connect a socket in the "+/power" column to row 7, column "a". Next, add a 330 Ohm resistor, placing the legs in row 7 column "b" and row 11 column "b". It may be necessary to bend the resistor to place the pins in sockets. Then add an LED with the long leg (anode) in row 11 column "c" and the short leg (cathode) in row 12 column "c". Finally, using a wire, connect row 12 column "a" with a socket in column "-/ground".

Students should trace the flow of electrons in their circuits, since that often helps discover mistakes in the placement of circuit components. Power should be provided to the circuit only after it has been built and checked. *This precaution decreases the number of LEDs that burn out because the circuits were constructed incorrectly.* To provide power, the microcontroller must be connected either to the 4 AA batteries in the power source holder or to the USB port of a computer. If the circuit is built properly, the LED will light up.

Understanding the potentiometer and the button - analog vs. digital:

Students learn how the potentiometer and button work and determine which is an analog and which is a digital device. Introduce students to the concept of a digital signal–one or two discrete values, such as ON and OFF–and contrast it with an analog signal, a sequence of continuous values. The students now build two different circuits (Figures 5 and 6) to experimentally determine if the button and potentiometer are digital or analog devices. In both circuits, the devices control an LED. The students should be encouraged to experiment by pushing the button and turning the potentiometer to various values and seeing what happens to the LED. Students should use their observations to decide whether the button and the potentiometer are analog or digital, and explain their reasoning. The button serves as a switch to turn the LED on and off, therefore the button is a digital device. In contrast, the potentiometer serves as a dimmer switch for an LED, making it an analog device.

Part 2. Microcontrollers: Combining hardware and software (~2-3 hrs)
In Part 2, students are introduced to using the microcontroller to control the LED. First, they learn to use the microcontroller to read in values from the button and the potentiometer and output these values to the computer. Next, they use software to control the button, potentiometer, and LED.

Becoming more familiar with the microcontroller: Because students will use the microcontroller in Part 2 more extensively than in Part 1, the instructor should point out the additional pins on the microcontroller that will be used in Part 2. These include the “Analog In” pins and the “Digital (~PWM)” section with pins “0-13”. (Figure 3). The “Analog In” pins are used to connect analog devices, such as a potentiometer, and read-in analog signals (but not output them). The “Digital (~PWM)” pins 0-13 are used to read-in and output digital signals. Importantly, the pins marked with a (~) are also able to output analog signals.

Learning about software/computer programs: Students will use computer programs in Part 2 to control the LED. The goal is to introduce students to programs and some of their characteristics.

The programs written for use with the Arduino microcontrollers are called “sketches”, and have syntax or rules. For example, each line in the sketch must end with a semicolon (;) names of variables, (which store values) should not have spaces or start with a number. In addition, it is good practice to annotate a sketch, meaning that explanations of each line are added. The “//” symbol identifies everything after it as a comment, not executable code.

Students should examine the sketches (see Supplementary Information, “Programming Basics”) to see several examples of these rules. For example, in the sketch called Programming Basics look at line:

```
int delay_time;    // delay time (in milliseconds)
```

Things to notice are: 1) the line of code ends with a semicolon, 2) the variable “delay_time” does not contain spaces or start with a number, and 3) “delay time (in milliseconds)” after the “//” symbol is the annotation that explains what is stored in the variable “delay_time”. In this line of the sketch, “int” refers to the integer data type used to store the number for the delay time.

The instructor should explain the basic organization of a sketch. See https://www.arduino.cc/en/Tutorial/Sketch.

Using sketches to read-in values from devices and determining the range of values: Students now learn how the values from digital and analog devices, the button and potentiometer, are read in

by the microcontroller and displayed on the computer. The students will determine the range of values generated by these devices.

Students should first build the circuit shown in Figure 7, but without the LED. The circuit includes a button and a potentiometer, which are connected to the microcontroller so that their values can be read-in and displayed on the computer.

Next, students should plug the USB connection into the computer and the microcontroller and open the file "Programming Basics". In the "Tools" menu, the proper Breadboard (Arduino/Genuino Uno) and Port (USB) should be selected. For the first part, we only want the button to be active, so students should make sure that only the button is on. To do this, the three lines of code that control which sensor is active should read:

```
const bool button_on = true;
const bool pot_on = false;
const bool led_on = false;
```

Now, the students are ready to upload their sketch to the Arduino microcontroller by clicking the right arrow button at the top left of the window. Next, to read in the value of the button, the students must open the Serial Monitor by clicking the magnifying glass in the top right corner of the window. It is important to make sure that the baud rate of 4800 is selected from the middle pull-down menu on the bottom of the newly opened page, because that is the value specified in the sketch. After a few seconds, the terminal window will display "Button value: 1".

Students should push the button and note the new read-in value of 0, displayed in the terminal window. They should also explain if it makes sense to see two values for the button, 0 and 1. Yes – on and off for a digital device.

Next, students should determine the range of read-in values for the potentiometer. The instructor should ask students to modify the code so that only the potentiometer is turned on. The three lines of code that control the sensor input should now read:

```
const bool button_on = false;
const bool pot_on = true;
const bool led_on = false;
```

The same steps should be followed as for the button, with the students twisting the knob on the potentiometer to change the value that is read in. The read-in values should range from 0 to 1023. Again, the students should explain whether or not it makes sense to see a range of values displayed. Yes, because the potentiometer is acting as an analog device.

Learning to control the LED via software, altering how frequently it blinks: Students are now ready to add the LED to their circuit (as shown in Figure 4). The blinking rate of the LED will be controlled through the microcontroller, through the "delay time" variable. First, in the sketch "Programming Basics" the instructor should ask students to change lines of code in the variable declaration section to turn the LED on (i.e. change "const bool led_on = false;" to "const bool led_on = true;"). After students upload the program, they should describe what they observe: the LED is on for half a second, then off for half a second, alternating in this time pattern.

This behavior occurs because the delay time has been set to 500 milliseconds ("delay_time = 500;"). The instructor should ask students to change the value of the delay time in the function

setup and describe their observations. Can they explain their observation by looking through the lines of code in the loop function that controls turning the LED on and off? The if statement following "// turn led off/on" states that if the LED is on ("if (led_on)") then it will be set to HIGH (5 Volts - on) for 500 milliseconds, and then to LOW (0 Volts - off), for 500 milliseconds. Therefore, changing the value of the delay time will change the blinking pattern of the LED such that it will now stay on and then off for the amount of time specified by the delay time.

Using sketches to generate and control voltage signals that will power the LED:
Using the circuit in Figure 7, students should upload the program "Controlling LED" and experiment with changing the values in the if statements (from "false" to "true" and vice versa). After each change, the students should describe their effects on the LED. The button can be activated by the line "const bool button_on = true;". Next, by setting different boolean values to "true", the LED can be controlled in different ways through the button. Only one boolean value should be set to "true" at any one time. The line "bool button_no_delay = true;" will turn the LED off as long as the button is pressed, "bool button_delay = true;" will turn the LED on for the length of the delay period after the button is pressed, and "bool button_state = true;" will switch the state of the LED from off to on, or vice versa, upon pressing the button.

In the case of the potentiometer, with "const bool pot_on = true;" the potentiometer is activated. "bool pot_delay = true;" will allow the LED blinking rate to be set by the potentiometer. If the potentiometer is set to 0 (by turning the knob), the LED will be on and then off for 1000 ms each. In contrast, the maximum potentiometer value of 1023 will cause the LED to be on and then off for 10 ms each. Within the range specified in the sketch, the potentiometer's value linearly maps to the delay time. With the potentiometer on ("bool pot_brightness = true;"), changing the value of the potentiometer will control the brightness of the LED. The "map" function converts the low and high values of the input range (i.e. 0 to 1023) to a specified output range (for example, 0 to 1,000 ms of blinking delay and duration).

Students should describe how different if statements control the LED, and alter lines of code to make the LED blink in specified ways - by changing the delay time value or changing the minimum or maximum brightness of the LED ("delay_time = map(pot_val, 0, 1023, X, Y);" or "brightness_val = map(pot_val, 0, 1023, X, Y);").

Part 3. Software-based circuit game (~1-2 hrs)

Building on Parts 1 and 2, in Part 3, students will combine circuits to build a game. The goal is for the students to play with the game and examine the code to explain how the game works. In the game, the LEDs will be controlled through the microcontroller using the sketch "Game" and will incorporate the button and potentiometer. If time is limited, the instructor may prefer to build the circuit shown in Figure 8 ahead of time. To play the game, the sketch "Game" will need to be uploaded.

Description of the game: The game lights up one LED at a time, but with a delay between them, to create the illusion of a light moving to the right along the strip of LEDs. The speed at which it moves is controlled by the potentiometer. The goal of the game is to press the button when the middle yellow LED is on. If the player is successful, the green LEDs will light up 3 times. If the player fails to press the button at the right time, the red LEDs light up 3. The game is automatically

restarted after the button is pressed and the LEDs flash 3 times. If the button is continuously pressed, whether the game is won or lost, the red LEDs will continue to flash.

Guiding students to determine how the game works: The instructor should ask questions such as: “How is the illusion of light moving along the strip of LEDs created?”, “What happens when you twist the potentiometer?”, “What happens when you push the button?”, and “Does it matter which color LED is lit up when you twist the potentiometer or push the button?”

Challenge exercises that modify the game: There are several possibilities to expand this part of the module to allow for student exploration. For example, the instructor should challenge students to modify parameters of the game. Students can change the number of LEDs incorporated into the game, or which LED needs to be lit when the button is pushed to win the game, or change the LED pattern that lights up after the player wins or loses, or change the direction in which the LEDs light up. To better understand the computer program, it may be helpful to reference the Arduino programming language functions and variables at https://www.arduino.cc/reference/en/.

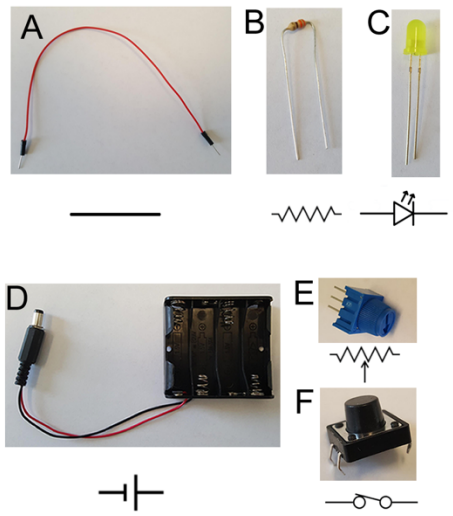

Figure 1. Circuit components with the corresponding schematics shown underneath each image. (A) A wire, covered by red insulating material and with exposed ends to form connections on the Breadboard and create a path for the electricity (Figure 2). (B) A resistor (with a resistance of 330 Ohms, indicated by the colored strips on the resistor), which reduces the flow of electrons in a circuit and releases energy as heat. (C) A yellow Light Emitting Diode (LED). The short leg is the “cathode/-” and the long leg is the “anode/+”. (D) A power source (shown without batteries included), which is the voltage source that enbles electricity to flow through the circuit. (E) Potentiometer (a variable resistor) showing three pins that connect it to the Breadboard and the twistable knob on top. The middle pin outputs voltage. (F) Button, which is a type of switch, with pins (three of the four show), used to connect it to the Breadboard. The schematic shows the button in the closed state (pressed), allowing for current flow.

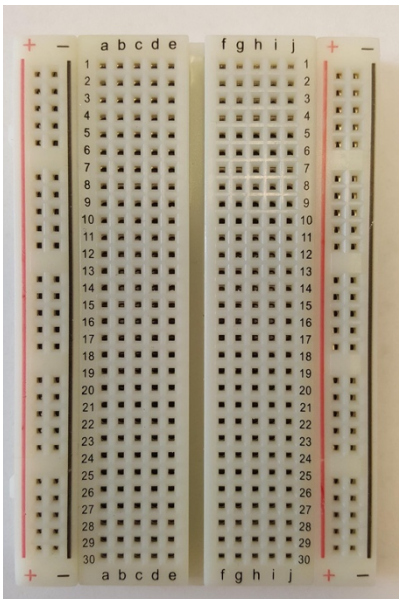

Figure 2. A breadboard, composed of 30 rows, numbered 1-30, and columns a-j in addition to a "+/power" (red line) and "-/ground" (black line) column on either side. Columns a-e and f-j are connected such that powering any socket a-e or f-j in a single row will power all sockets a-e or f-j, respectively, in that row. Once power is routed to the "+/power" (red line) column, all sockets will have power. Similarly, once one socket is grounded in the "-/ground" (black line) column, all sockets can be used as ground.

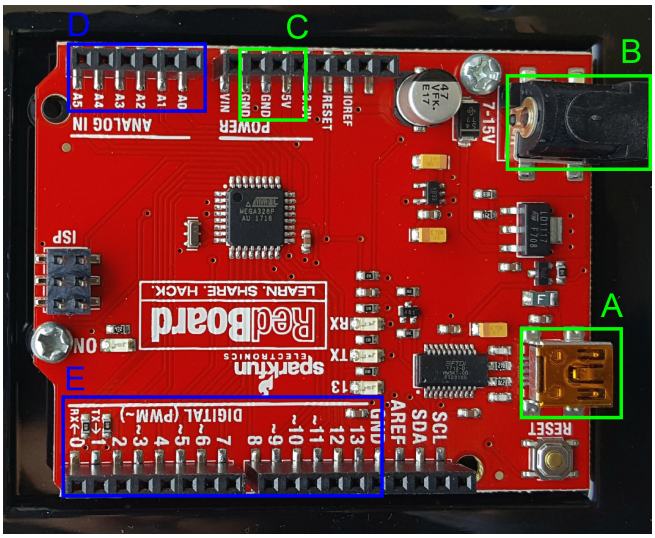

Figure 3. The Arduino microcontroller. Parts of the microcontroller that are used in Part 1 of the module are boxed in green, while those used in Parts 2 and 3 are boxed in blue. (A) Plug for USB connector to connect the microcontroller to a computer. (B) Battery pack connector. (C) Pins within the "Power" portion of the microcontroller that will be used. "GND" are ground pins and "5V"

is the pin used to connect a 5V power source. (D) Five pins within the “Analog In” portion of the microcontroller can be used to connect analog devices to the microcontroller. (E) Pins within the “Digital (PWM~)” portion of the microcontroller used to read-in and output digital signals.

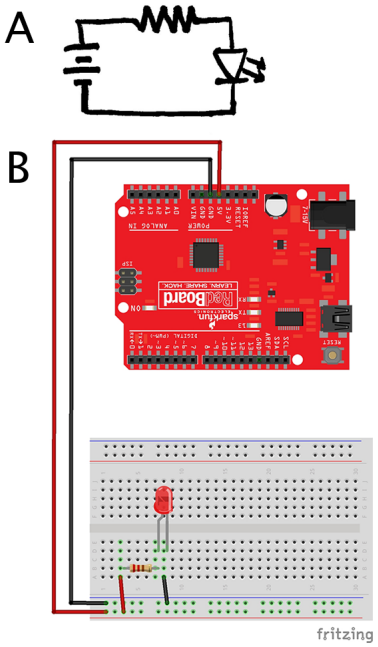

Figure 4. Building a simple circuit to illuminate an LED. (A) The schematic of the circuit. (B) The circuit shown using the breadboard view (made using Fritzing, available for download at fritzing.org). Ground (black wire) and power (red wire) connect the Arduino microcontroller (red rectangle) to the breadboard (grey rectangle). Resistors, potentiometer, button, and LEDs are placed as shown. The green lines indicate current flow. For this circuit, the only purpose of the microcontroller is to provide power (5V) to the breadboard. The microcontroller itself is powered by connecting it to a computer via the USB cable or connecting it to the battery pack that requires four AA batteries. Note that the LEDs need to be connected in a particular direction. The anode is the positive (+) end and has a longer leg. The cathode is the negative (-) end and has a shorter leg.

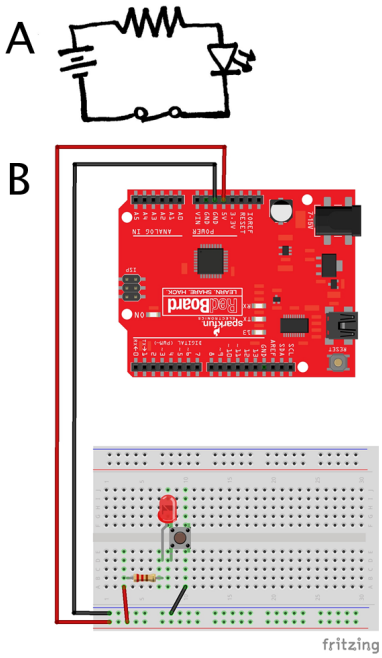

Figure 5. Building a circuit with a button and an LED. (A) The schematic of the circuit. (B) A circuit, shown using the breadboard view (made using Fritzing, available for download at fritzing.org), with a button (gray square with brown circle), resistor (tan rod), and an LED.

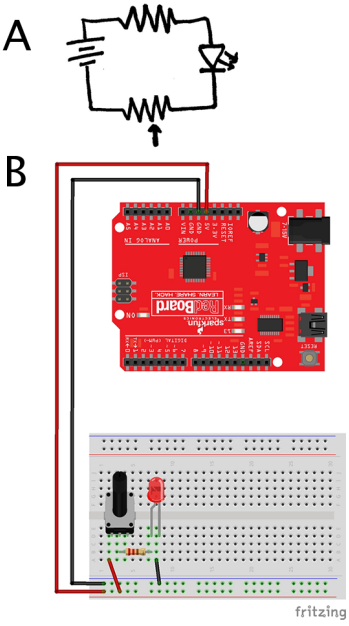

Figure 6. A circuit with a potentiometer (grey square with black knob), resistor (tan rod), and an LED (red). (Image made using Fritzing, available for download at fritzing.org)

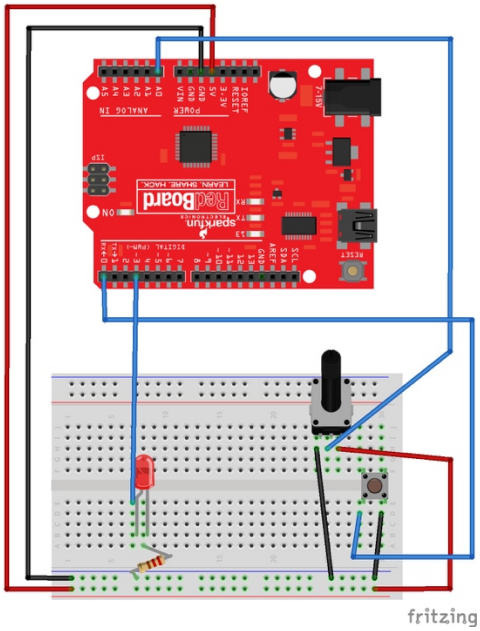

Figure 7. Circuits used in “Programming Basics” and “Controlling LED” sketches, with a potentiometer (grey square with black knob), an LED, a resistor, and a button (gray square with brown circle). (Image made using Fritzing, available for download at fritzing.org)

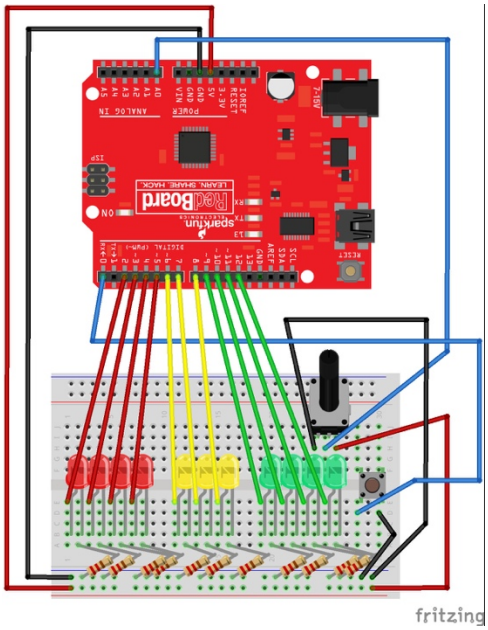

Figure 8. The circuit diagram for the LED game, which features four red, three yellow, and four green LEDs, a resistor (tan rod) for each LED, a potentiometer (grey square with black knob) and a button (grey square with brown circle). (Image made using Fritzing, available for download at fritzing.org)

Acknowledgements: The authors are greatful for support from the Raymond and Beverly Sackler Institute for Biological, Physical and Engineering Sciences at Yale University, NIH National Insitute of Biomedical Imaging and Bioengineering Grant No. T32EB019941, NSF Grant Nos. DBI-1458609 and DBI-1755494, and NIH National Cancer Institute Grant No. U54CA209992. The authors thank Pathfinder Hopkins School, including Director Michael Van Leesten and Jennifer Lane for their longstanding collaboration and Nadia Abascal for her critical reading of the manuscript.

Table S1.

The name of each item, its description, and cost are listed. One of each item is required for each group of 2 to 4 students.

Name of Item	Description	Supplier	Cost
SparkFun Thinker Kit (https://www.sparkfun.com/products/14556)	Arduino microcontroller, called 'RedBoard', with a voltage source (no soldering required), breadboard, potentiometer, different types of resistors, a button, LED bulbs, and other accessories, along with a detailed guide explaining how to build circuits. Free software interface is also provided using the Arduino programming language.	SparkFun	$49.95
Extra LEDs (https://www.sparkfun.com/products/12062)	Various colored LEDs, supplied in a pack of 20.	SparkFun	$2.95
Computer (a basic laptop/desktop is adequate)	For Part 1, needed as a power source. For Parts 2 and 3, needed to manipulate code.	N/A	Ideally provided by institution
4 AA Batteries	Only needed if using battery holder to power microcontroller in Part 1.	amazon	$3.59

Sketch: Programming Basics

```
// global scope

const int led_pin = 3;    // pin used for the led (digital output)
const int button_pin = 0; // pin used for the button (digital input)
const int pot_pin = 0;    // pin used for the potentiometer (analog input)
int led_val, pot_val, button_val;  // variables to hold the value for the led/pot/button
int delay_time;           // delay time (in milliseconds)

// Boolean values to control which sensor is active
const bool button_on = true;
const bool pot_on = false;
const bool led_on = false;

void setup() {
  // set up digital pins
  // the words highlighted in orange are built-in commands that the microcontroller knows
  pinMode(button_pin, INPUT_PULLUP);  // sets pin to 5V when circuit is open, 0V when closed
  pinMode(led_pin, OUTPUT);           // can ouput 0V or 5V
```

```
  // no need to set up analog pins since
  // they are only configured for input

  // enable data trasmission to laptop
  // (click on the magnifying lens icon on the upper right-hand corner)
  Serial.begin(4800);

  // choose a delay time (in milliseconds)
  delay_time = 500;
}

void loop() {
  // display value of button
  if (button_on) {                    // this code is executed if button_on is set to 'true'
    button_val = digitalRead(button_pin);   // pin reads in the current digital value (HIGH (1)/ LOW
(0)) and assigns it to 'button_val'
    Serial.print("Button value: ");         // prints the button value
    Serial.println(button_val);
    delay(delay_time);                      // program pauses for 'delay_time' milliseconds
  }

  // display value of potentiometer
  if (pot_on) {
    pot_val = analogRead(pot_pin);          // pin reads in the current analog value (0 - 1023) and
assigns it to 'pot_pin'
    Serial.print("Potentiometer value: ");
    Serial.println(pot_val);
    delay(delay_time);
  }

  // turn led off/on
  if (led_on) {
    digitalWrite(led_pin, HIGH);            // sets led_pin to 'HIGH' (5V)
    delay(delay_time);
    digitalWrite(led_pin, LOW);             // sets led_pin to 'LOW' (0V)
    delay(delay_time);
  }
}
```

Sketch: Controlling LED

```
// global scope
const int led_pin = 3;    // pin used for the led (digital output)
const int button_pin = 0; // pin used for the button (digital input)
const int pot_pin = 0;    // pin used for the potentiometer (analog input)
int led_val, pot_val, button_val;  // variables to hold the value for the led/pot/button
int delay_time;           // delay time (in milliseconds)
```

```
int brightness_val;

// Boolean values to control which sensor is active
const bool button_on = false;
bool button_no_delay = false;
bool button_delay = false;
bool button_state = true;

const bool pot_on = true;
bool pot_delay = false;
bool pot_brightness = true;

// values for light switch module
bool switch_state = false;
int button_val_prev = digitalRead(button_pin);
int counter = 0;

void setup() {
  // set up digital pins
  pinMode(button_pin, INPUT_PULLUP);  // sets pin to 5V when circuit is open, 0V when closed
  pinMode(led_pin, OUTPUT);           // can ouput 0V or 5V

  // no need to set up analog pins since
  // they are only configured for input

  // enable data trasmission to laptop
  Serial.begin(4800);

  // choose a delay time (in milliseconds)
  delay_time = 2000;
}

void loop() {
  // control led (on/off) with a button
  if (button_on) {                      // this code is executed if button_on is set to 'true'

    // read in button value
    button_val = digitalRead(button_pin);   // pin reads in the current digital value (HIGH (1)/ LOW
(0)) and assigns it to 'button_val'

    // control led (with no delay)
    if (button_no_delay) {
      if (button_val) {                 // led is on until button is pressed
        //if (!button_val) {              // led is off until button is pressed
        digitalWrite(led_pin, HIGH);        // sets led_pin to 'HIGH' (5V)
      } else {
        digitalWrite(led_pin, LOW);         // sets led_pin to 'LOW' (0V)
      }

      Serial.print("Button value: ");         // prints the button value
      Serial.println(button_val);
```

```
  }

  // control led (with a delay)
  if (button_delay) {
    if (!button_val) {
      digitalWrite(led_pin, HIGH);        // sets led_pin to 'HIGH' (5V)
      delay(delay_time);
    } else {
      digitalWrite(led_pin, LOW);         // sets led_pin to 'LOW' (0V)
    }

    Serial.print("Button value: ");         // prints the button value
    Serial.println(button_val);
  }

  // control led (state switch)
  if (button_state) {
    // flip the switch_state by detecting a button press (press and release)
    if (button_val != button_val_prev) {
      button_val_prev = button_val;
      counter++;
    }
    if (counter == 2) {
      switch_state = !switch_state;
      counter = 0;
    }

    // prints the button value
    //Serial.print("Counter value: ");
    //Serial.println(counter);
    Serial.print("switch_state value: ");
    Serial.println(switch_state);

    // flip the led on/off depending on the switch_state
    if (switch_state) {
      digitalWrite(led_pin, HIGH);
    } else {
      digitalWrite(led_pin, LOW);
    }
  }
}


 // control led with a potentiometer
 if (pot_on) {

   // read in pot value
   pot_val = analogRead(pot_pin);          // pin reads in the current analog value (0 - 1023) and
assigns it to 'pot_pin'
   Serial.print("Potentiometer value: ");
   Serial.println(pot_val);
```

```
    // control led blinking delay
    if (pot_delay) {
      // Map 'pot_val' linearly to 'delay_time'
      delay_time = map(pot_val, 0, 1023, 1000, 10);
      Serial.print("delay_time: ");
      Serial.println(delay_time);

      digitalWrite(led_pin, HIGH);          // sets led_pin to 'HIGH' (5V)
      delay(delay_time);
      digitalWrite(led_pin, LOW);           // sets led_pin to 'LOW' (0V)
      delay(delay_time);
    }

    // control led brightness via pulse-width modulation (PWD)
    if (pot_brightness) {
      brightness_val = map(pot_val, 0, 1023, 0, 255);  // operates on the range of 0 - 255
      analogWrite(led_pin, brightness_val);
      Serial.print("brightness_val: ");
      Serial.println(brightness_val);
    }
  }
}
```

Sketch: Game

```
// pins used for the button and potentiometer
const int button_pin = 0;
const int pot_pin = 0;
int led_pin_win, led_pin, led_pin_start, led_start, led_end, pot_val, button_state;
bool game_state, game_won, stripe_left, stripe_right;
float counter;

void setup() {
  // set up digital pins
  pinMode(button_pin, INPUT_PULLUP);
  for (led_pin = 2; led_pin < 13; led_pin++) {
    pinMode(led_pin, OUTPUT);
  }

  stripe_left = true;
  stripe_right = !stripe_left;

  button_state = 1;
  game_state = true;
  led_pin_win = 7;
  led_pin_start = 1;
  led_pin = led_pin_start;
  counter = 0;
}
```

```
void loop() {
  // single stripe moving to left/right
  if (true) {
game_start:
    if (game_state) {
      // read pot value to adjust led delay
      pot_val = analogRead(pot_pin)+200;
      pot_val = map(pot_val, 0, 1023, 1000, 10);
      //pot_val = map(pot_val, 0, 1023, 1000, 10) + 5*sin(counter);
      //counter = counter+0.1;
      pot_val = (pot_val < 1)?1:pot_val;

      // read button (to prevent cheating)
      button_state = digitalRead(button_pin);
      if (button_state == 0) {
        game_state = false;
        game_won = false;
        goto game_over;
      }

      // turn led on/off
      digitalWrite(led_pin, HIGH);
      unsigned long previous_millis = millis();
      while ((millis() - previous_millis) < pot_val) {
        // read button
        button_state = digitalRead(button_pin);
        if (button_state == 0) {
          game_state = false;
          if (led_pin == led_pin_win) {
            game_won = true;
            goto game_over;
          } else {
            game_won = false;
            goto game_over;
          }
        }
      }
      digitalWrite(led_pin, LOW);

      // move stripe left
      if (stripe_left) {
        if (led_pin < 13) {
          led_pin++;
        } else {
          led_pin = 1;
        }
      }

      // move stripe right
```

```
      if (stripe_right) {
        if (led_pin > 0) {
          led_pin--;
        } else {
          led_pin = 12;
        }
      }
    }

game_over:
    if (game_state == false) {
      for (int i = 0; i < 3; i++) {
        // select winning/losing lights
        if (game_won) {
          led_start = 9;
          led_end = 12;
        } else {
          led_start = 2;
          led_end = 5;
        }

        int led;
        for (led = led_start; led <= led_end; led++) {
          digitalWrite(led, HIGH);
        }
        delay(500);
        for (led = led_start; led <= led_end; led++) {
          if (led != led_pin) {
            digitalWrite(led, LOW);
          }
        }
        delay(500);
      }

      // restart game
      game_state = true;
      digitalWrite(led_pin, LOW);
      led_pin = led_pin_start;
      goto game_start;
    }
  }
}
```