Table of Contents

SparkFun Inventor's Kit Teacher's Helper

These worksheets and handouts are supplemental material intended to make the educator's job a little easier by providing easily edited content. We would never tell you how to teach this technology, we just want to make your job easier by giving you the tools to incorporate this technology into your curriculum. You can use these files however you see fit. Add a question here, delete a question there and definitely add some graphics if you like. The worksheets are intended for use after completion of the SparkFun Inventor's Kit or as you go along. There is no particular order so you can use whichever worksheets you wish, whenever you think is best.

The “Middle School” worksheet contents may be considered advanced or difficult by many standards, simply because SparkFun believes in challenging learners and presenting lots of information in the hopes that the users will surpass our expectations. This kit is also just the very beginning to the world of physical computing, it establishes the basic skills you and your students will need to continue to explore and create.

The SparkFun Inventor's Kit is a great introductory tool to get people interested in electronics and physical computing. This is the first collection of worksheets that pertain to the S.I.K.. We would appreciate any feedback you feel would be useful. Topics we missed, projects you put together using the SparkFun Inventor's Kit, typos, gripes, material you have put together about this type of technology that you would like to share or stuff that was really, really useful. Basically we want you and your students to eventually be able to build robots that can sing, dance and take over the world... or at least your imagination. And send us some pictures while you're at it, huh?

Included material:
- Worksheets and handouts for S.I.K. circuits 1 – 14
- Answers to worksheets (Creative answers left blank, Ohm's Law answers may vary a little)
- Expansion code for use in Arduino
- Images to create your own schematics
- Surveys for teachers and students to aid in development of material
- Fritzing (Virtual Prototyping Software)

This material is a work in progress. We invite you to be a part of creating and proofing the documentation, for example we need a worksheet on current, so feel free to contribute and expand this collection of material if you feel so inclined.

Send feedback, worksheets or completed surveys to: EdMaterials@Sparkfun.com

or snail mail: Attention Lindsay Craig
6175 Longbow Drive
Boulder, Colorado 80301

Material by: Lindsay Craig, Jim “The Engineer” Lindblom, and Ben Leduc-Mills
Images by: Dave Stadler and Lindsay Craig
Proofed by: Ben Leduc-Mills, Lindsay Craig, Jim “The Engineer” Lindblom, Chris “Cmac” McGrady, Michelle Shorter, Toni Klopfenstein, SparkFun's IT department and other equally awesome people.

Insert Attribution Here:

SIK Worksheets v.1.0
Getting started with Arduino

Installing Arduino

Mac platform	PC platform
1. Double click the file **arduino-0022.dmg** inside the folder \SIK Applications\Mac\ 2. Go to “Arduino” in the devices section of the finder and move the “Arduino” application to the “Applications” folder. 3. Go to the “Arduino” device, double click and install: “**FTDI drivers for Intel Macs 0022.pkg**” or “**FTDI drivers for PPC Macs 0022.pkg**” then Restart your computer. 4. Plug your Arduino board into a free USB port using the USB cord provided.	1. Unzip the file **arduino-0022** inside the folder \SIK Applications\PC\. We recommend unzipping to your c:\Program Files\ directory. 2. Open the folder containing your unzipped Arduino files and create a shortcut to Arduino.exe. Place this on your desktop for easy access. 3. Plug your Arduino board into a free USB port using the USB cord provided. Wait for a pop up box about installing drivers. 4. Skip “searching the internet”. Click “Install from a list or specific location” in the advanced section. Choose the location c:\program files\arduino-0022\drivers\Arduino Uno\ **(You may have to do this last step more than once)** **(If you are using the Duemilanove you will have to choose the sub-directory, FTDI USB Drivers and you will have to do this twice)**

If you are having issues with Java make sure you have the latest version of Java installed. Otherwise....
Ta da! You're ready to open the Arduino programming environment.

(For Linux info go to http://ardx.org/LINU)

Installing Fritzing

Mac platform	PC platform
1. Move the Fritzing folder from \SIK Applications\Mac\ to somewhere convenient on your computer. 2. Double click the file: **fritzing.2010.09.30.mac**	1. Move the Fritzing file from \SIK Applications\PC\ to somewhere convenient on your computer. 2. Double click the file: **fritzing.2010.09.30.pc**

Ta da! You're ready to start using Fritzing for virtual prototyping.

A few notes about Arduino setup

Selecting your board:

You are using the Arduino Uno board with an ATmega 328 micro-controller. This means you will need to select “Arduino Uno” as your board. To do this you click on the “Tools” menu tab, then click the “Board” tab and select “Arduino Uno”. If you are using a different board you will need to select the correct model in order to properly upload to your board.

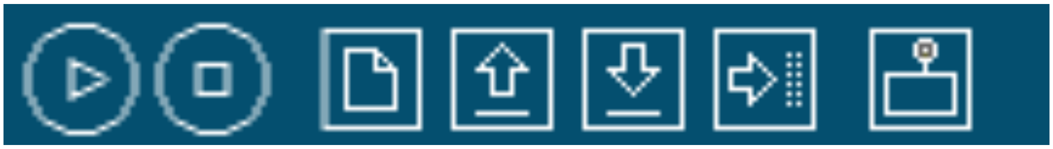
Selecting your com port:

Another option that is necessary to change occasionally is your “Serial Port”. This can also be found under the “Tools” menu tab. When you click on this tab you should be presented with at least one com port labeled “COM1” (or “COM2, etc....) This indicates which USB port your board is plugged into. Sometimes you will need to make sure you are using the correct com ports for your Arduino, here is some information on your com ports depending on which platform you are using:

Mac	PC
The Mac version of Arduino refreshes your com port list every time you plug in a device. For this reason all you really need to do is select the com port called “/dev/cu.usbserial-XXXX” where XXXX will be a value that changes.	The PC version of Arduino creates a new com port for every distinct board you plug into your computer. You will need to find out which com port is the board you are currently trying to use. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board. Reconnect the board and select that serial port.

A few more tidbits that will help to know....

There are seven buttons that look like this at the top of your Arduino window:



Here is what they do from left to right:

Compile	Stop	New	Open	Save	Upload	Serial Monitor
This checks your code for errors.	This stops the program.	This creates a new sketch.	This opens an existing sketch.	This saves the open sketch.	This uploads the sketch to your board.	Used to display Serial Communication.

For any words or phrases you are unfamiliar with try the Glossary in the back.

Basic Arduino Reference Sheet

Installation:

Arduino: http://www.arduino.cc/en/Guide/HomePage
Fritzing: http://fritzing.org/download/

Support:

Arduino: http://www.arduino.cc, http://www.freeduino.org, google.com
Fritzing: http://www.fritzing.org/learning/

Forums:

Arduino: http://forum.sparkfun.com/viewforum.php?f=32
Fritzing: http://fritzing.org/forum/

Basic Arduino code definitions:

setup( ): A function present in every Arduino sketch. Run once before the loop( ) function. Often used to set pinmode to input or output. The setup( ) function looks like:

```
void setup( ){
     //code goes here
     }
```

loop( ): A function present in every single Arduino sketch. This code happens over and over again. The loop( ) is where (almost) everything happens. The one exception to this is setup( ) and variable declaration. ModKit uses another type of loop called "forever( )" which executes over Serial. The loop( ) function looks like:

```
void loop( ) {
     //code goes here
     }
```

input: A pin mode that intakes information.

output: A pin mode that sends information.

HIGH: Electrical signal present (5V for Uno). Also ON or True in boolean logic.

LOW: No electrical signal present (0V). Also OFF or False in boolean logic.

digitalRead: Get a HIGH or LOW reading from a pin already declared as an input.

digitalWrite: Assign a HIGH or LOW value to a pin already declared as an output.

analogRead: Get a value between or including 0 (LOW) and 1023 (HIGH). This allows you to get readings from analog sensors or interfaces that have more than two states.

analogWrite: Assign a value between or including 0 (LOW) and 255 (HIGH). This allows you to set output to a PWM value instead of just HIGH or LOW.

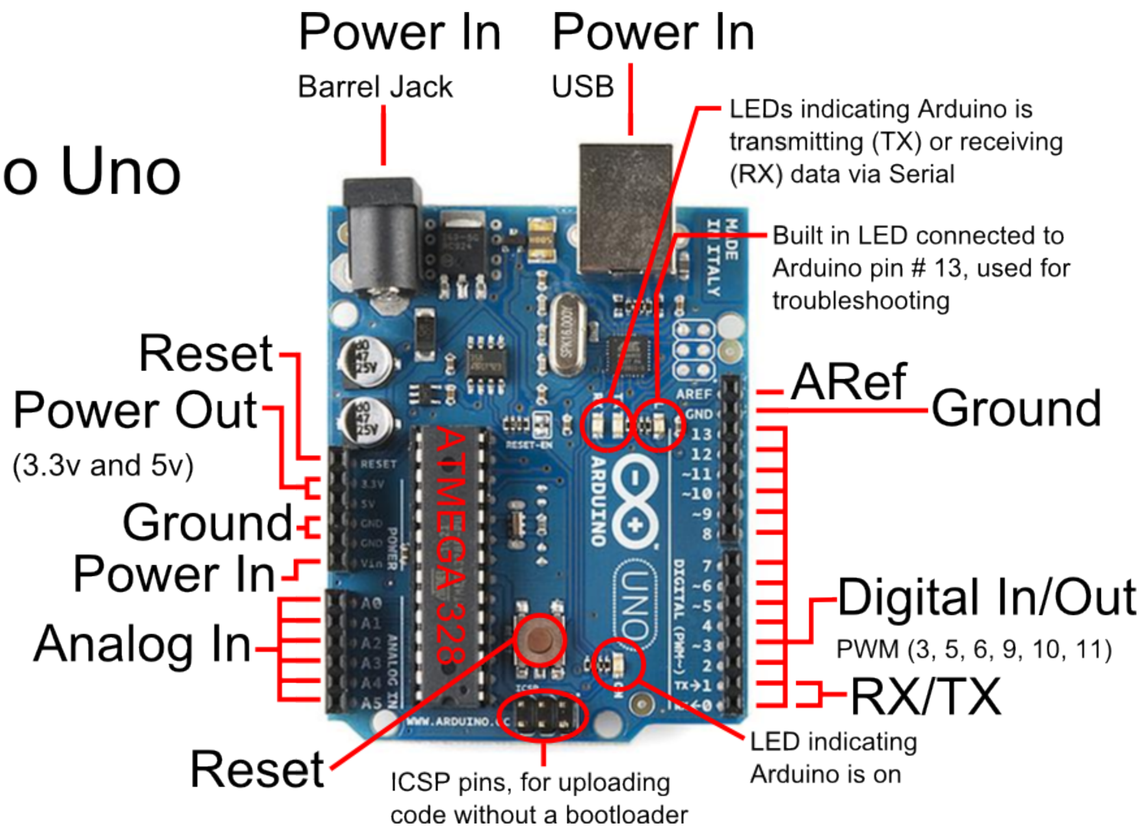
PWM: Stands for Pulse-Width Modulation, a method of emulating an analog signal through a digital pin. A value between or including 0 and 255. Used with analogWrite.

Arduino Uno pin type definitions: (Take a look at your Arduino board)

Reset	3v3	5v	Gnd	Vin	Analog In	RX/TX	Digital	PWM(~)	AREF
Resets Arduino sketch on board	3.3 volts in and out	5 volts in and out	Ground	Voltage in for sources over 7V (9V - 12V)	Analog inputs, can also be used as Digital	Serial comm. Receive and Transmit	Input or output, HIGH or LOW	Digital pins with output option of PWM	External reference voltage used for analog

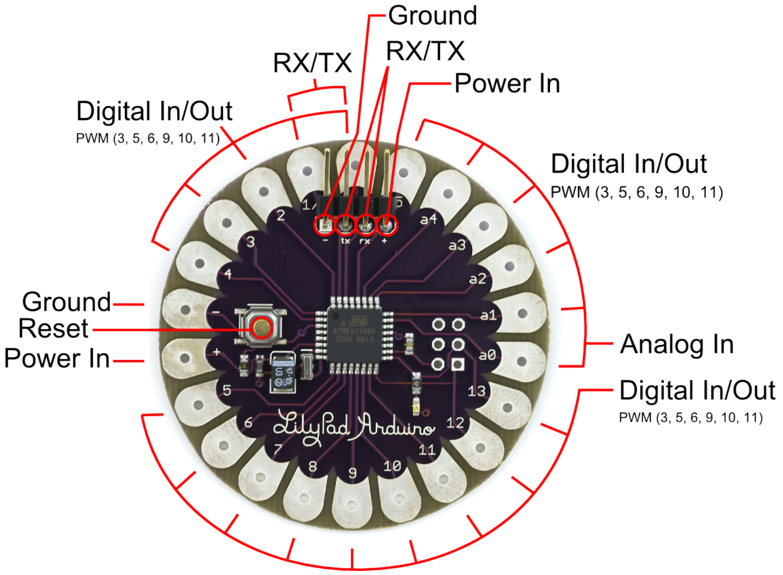
Basic Arduino Pin Reference Sheet

Arduino Uno

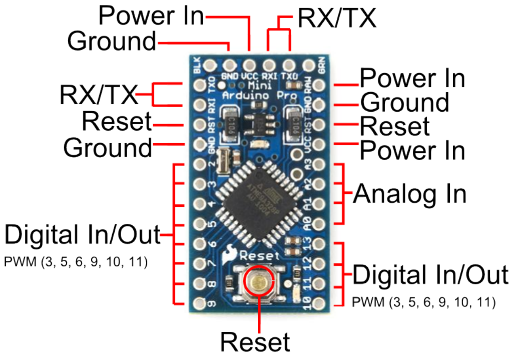

Power In
Barrel Jack
Power In
USB
LEDs indicating Arduino is transmitting (TX) or receiving (RX) data via Serial
Built in LED connected to Arduino pin # 13, used for troubleshooting
ARef
Ground
Reset
Power Out
(3.3v and 5v)
Ground
Power In
Analog In
ATMEGA 328
Digital In/Out
PWM (3, 5, 6, 9, 10, 11)
RX/TX
Reset
ICSP pins, for uploading code without a bootloader
LED indicating Arduino is on

These boards below use the same micro-controller, just in a different package. The Lilypad is designed for use with conductive thread instead of wire and the Arduino Mini is simply a smaller package without the USB, Barrel Jack and Power Outs. Other boards in the Arduino family can be found at http://arduino.cc/en/Main/Hardware.

Arduino Lilypad

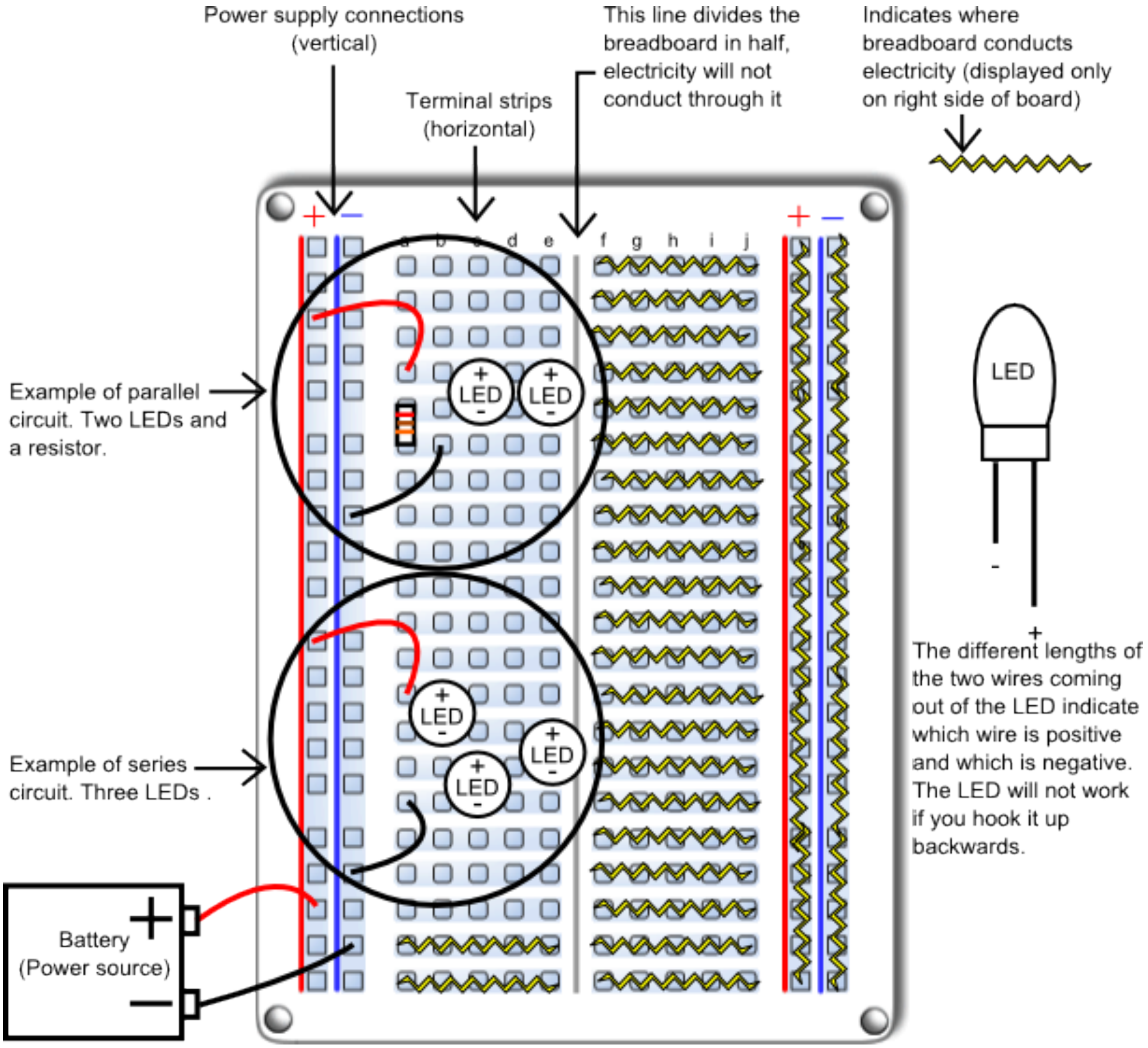


Arduino Mini



SIK Worksheets v.1.0 **Name:**
Breadboard **Date:**

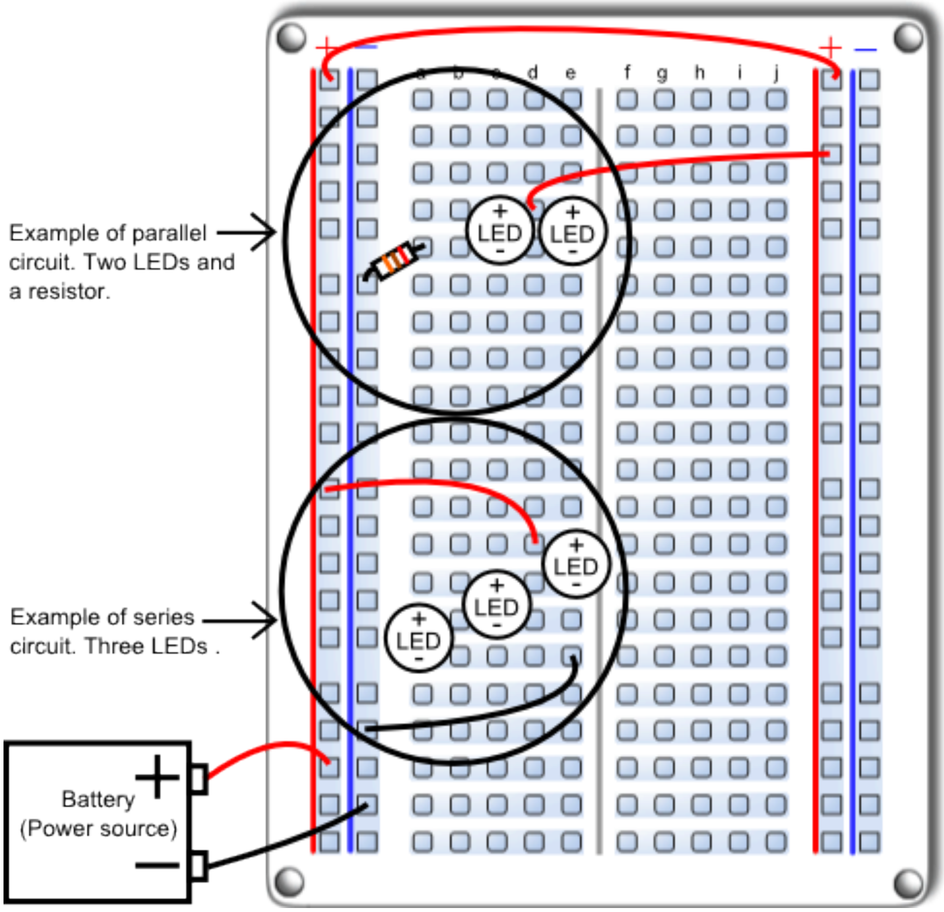
One of the most important tools for electrical prototyping and invention is the **breadboard**. It's not a piece of bread that you stick electronics into, it's a piece of plastic with holes to place wires into and copper connecting the holes so electricity can get to all the pieces you are working with. But not all the holes are connected! Below is a diagram and explanation of how a **breadboard** works as well as examples of parallel and series circuits. Not sure what parallel and series circuits are? Don't worry! The important thing is learning how to use the **breadboard** so you can play around with some electronics.


Power supply connections (vertical)
Terminal strips (horizontal)
This line divides the breadboard in half, electricity will not conduct through it
Indicates where breadboard conducts electricity (displayed only on right side of board)
+
–
a b c d e f g h i j
+
–
Example of parallel circuit. Two LEDs and a resistor.
LED
LED
Example of series circuit. Three LEDs .
LED
LED
LED
Battery (Power source)
+
–
LED
-
+
The different lengths of the two wires coming out of the LED indicate which wire is positive and which is negative. The LED will not work if you hook it up backwards.

The right side of this **breadboard** shows you which holes are connected and allow electricity to flow between them without anything else connecting them. This is made possible by strips of copper on the underside of the board. The **power** supply connections have a + and – indicating how to hook up your power source. The connections for the **power** supply run up and down. The **terminal** strips are labeled “a” through “j”, these connections run across the board, but are broken down the middle. This cuts the connection across the entire **terminal** area in half, giving you two unconnected sections to work with.

SIK Worksheets v.1.0 **Name:**
Breadboard **Date:**

Because the copper plating below the **power** supply connections and the **terminal** connections conduct electricity there are many different ways to hook up the same circuit and make it work. All that matters is that the electricity can flow through the entire circuit from power (+) to ground (-).


Example of parallel circuit. Two LEDs and a resistor.
Example of series circuit. Three LEDs .
Battery (Power source)
+
—
a b c d e
f g h i j
LED
+
-

This is an example of the same two circuits from the previous page hooked up in different ways that still work the same. There are many differences between this picture and the previous one. First of all, at the very top there is a wire connecting the **positive (+) power** terminal on the left with the **positive (+) power** terminal on the right. Now it is possible to supply power to your circuits from either side of the board. That's why this example of a parallel circuit has a red wire stretching all the way to the **positive (+) power** terminal on the right side. What would you do if you wanted to use the **ground (-) power** terminal on the right side of the board?

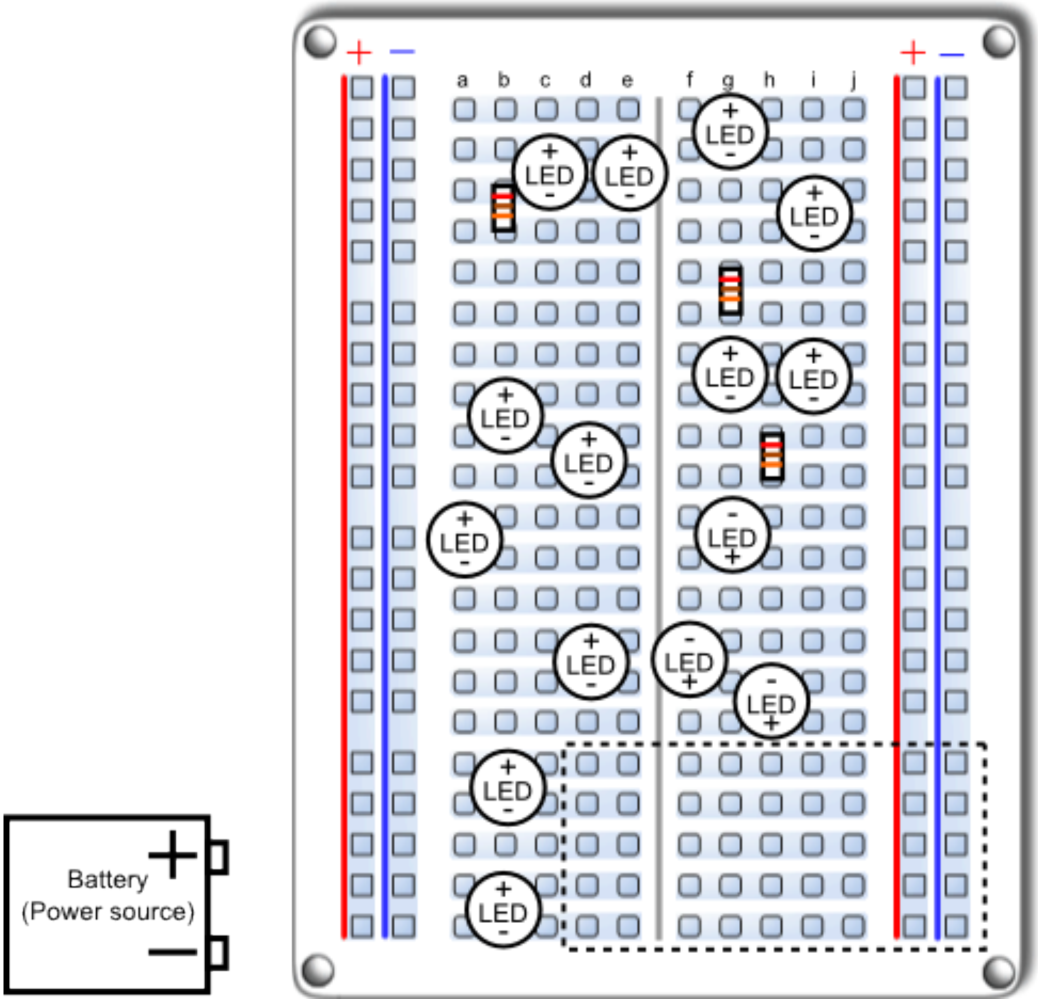
Another thing that has changed in the parallel circuit example is the position of the resistor. In the previous image, the resistor went from the row with the negative LED connections to the row below with a wire connecting that final row to the **ground (-) power** terminal. This is an example of tossing out a wire because you can use the wires coming out of components to plug directly into the **power** terminals. You can do the same thing with your LEDs if you like, try it out.

Lastly, the positions of the LEDs have changed in both examples. This is because it doesn't matter where the LEDs are positioned to the left or right (columns “a” through “e”), as long as they are plugged into the correct rows (up and down).

SIK Worksheets v.1.0 **Name:**
Breadboard **Date:**

Here are some quick questions to make sure you understand the Breadboard.

1. Circle the **power** terminals below, make sure you get all of them.
2. Draw wires to complete six LED circuits that will work. Each circuit needs either a total of three LEDs or a resistor. Use all **power** terminals at least once and don't forget to hook up your battery.
3. Inside the dotted lines draw lines to show where electricity will conduct without plugging anything else in.



You may be wondering why this piece of equipment is called a breadboard. It's not because the board goes well with butter or jam, it is because the board looks like the type of boards people used to lay bread on to cool.

SIK Worksheets v.1.0 **Name:**
Analog **Date:**

All of the electrical signals that the Arduino works with are either Analog or Digital. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

Analog

A continuous stream of information with values between and including 0% and 100%.

Humans perceive the world in analog. Everything we see and hear is a continuous transmission of information to our senses. The temperatures we perceive are never 100% hot or 100% cold, they are constantly changing between our ranges of acceptable temperatures. (And if they are out of our range of acceptable temperatures then what are we doing there?) This continuous stream is what defines analog data. Digital information, the complementary concept to Analog, estimates analog data using only ones and zeros.

In the world of Arduino an Analog signal is simply a signal that can be HIGH (on), LOW (off) or anything in between these two states. This means an Analog signal has a voltage value that can be anything between 0V and 5V (unless you mess with the Analog Reference pin). Analog allows you to send output or receive input about devices that run at percentages as well as on and off. The Arduino does this by sampling the voltage signal sent to these pins and comparing it to a voltage reference signal (5V). Depending on the voltage of the Analog signal when compared to the Analog Reference signal the Arduino then assigns a numerical value to the signal somewhere between 0 (0%) and 1023 (100%). The digital system of the Arduino can then use this number in calculations and sketches.

To receive Analog Input the Arduino uses Analog pins # 0 - # 5. These pins are designed for use with components that output Analog information and can be used for Analog Input. There is no setup necessary, and to read them use the command:

```
analogRead(pinNumber);
```

where pinNumber is the Analog In pin to which the the Analog component is connected. The analogRead command will return a number including or between 0 and 1023.

The Arduino also has the capability to output a digital signal that acts as an Analog signal, this signal is called Pulse Width Modulation (PWM). Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. To output a PWM signal use the command:

```
analogWrite(pinNumber, value);
```

where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). On the Arduino UNO PWM pins are signified by a ~ sign. For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

Examples of Analog:
Values: Temperature, volume level, speed, time, light, tide level, spiciness, the list goes on....
Sensors: Temperature sensor, Photoresistor, Microphone, Turntable, Speedometer, etc....

Things to remember about Analog:

Analog Input uses the Analog In pins, Analog Output uses the PWM pins
To receive an Analog signal use: `analogRead(pinNumber);`
To send a PWM signal use: `analogWrite(pinNumber, value);`
Analog Input values range from 0 to 1023 (1024 values because it uses 10 bits, 2^{10})
PWM Output values range from 0 to 255 (256 values because it uses 8 bits, 2^8)

SIK Worksheets v.1.0 **Name:**

Digital **Date:**

All of the electrical signals that the Arduino works with are either Analog or Digital. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

Digital

An electronic signal transmitted as binary code that can be either the presence or absence of current, high and low voltages or short pulses at a particular frequency.

Humans perceive the world in analog, but robots, computers and circuits use Digital. A digital signal is a signal that has only two states. These states can vary depending on the signal, but simply defined the states are ON or OFF, never in between.

In the world of Arduino, Digital signals are used for everything with the exception of Analog Input. Depending on the voltage of the Arduino the ON or HIGH of the Digital signal will be equal to the system voltage, while the OFF or LOW signal will always equal 0V. This is a fancy way of saying that on a 5V Arduino the HIGH signals will be a little under 5V and on a 3.3V Arduino the HIGH signals will be a little under 3.3V.

To receive or send Digital signals the Arduino uses Digital pins # 0 - # 13. You may also setup your Analog In pins to act as Digital pins. To set up Analog In pins as Digital pins use the command:

```
pinMode(pinNumber, value);
```

where pinNumber is an Analog pin (A0 – A5) and value is either INPUT or OUTPUT. To setup Digital pins use the same command but reference a Digital pin for pinNumber instead of an Analog In pin. Digital pins default as input, so really you only need to set them to OUTPUT in pinMode. To read these pins use the command:

```
digitalRead(pinNumber);
```

where pinNumber is the Digital pin to which the Digital component is connected. The digitalRead command will return either a HIGH or a LOW signal. To send a Digital signal to a pin use the command:

```
digitalWrite(pinNumber, value);
```

where pinNumber is the number of the pin sending the signal and value is either HIGH or LOW.

The Arduino also has the capability to output a Digital signal that acts as an Analog signal, this signal is called Pulse Width Modulation (PWM). Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. To output a PWM signal use the command:

```
analogWrite(pinNumber, value);
```

where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

Examples of Digital:

Values: On/Off, Men's room/Women's room, pregnancy, consciousness, the list goes on....

Sensors/Interfaces: Buttons, Switches, Relays, CDs, etc....

Things to remember about Digital:

Digital Input/Output uses the Digital pins, but Analog In pins can be used as Digital

To receive a Digital signal use: `digitalRead(pinNumber);`

To send a Digital signal use: `digitalWrite(pinNumber, value);`

Digital Input and Output are always either HIGH or LOW

SIK Worksheets v.1.0 **Name:**
Input **Date:**

All of the electrical signals that the Arduino works with are either input or output. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

Input Signals

A signal entering an electrical system, in this case a micro-controller. Input to the Arduino pins can come in one of two forms; Analog Input or Digital Input.

Analog Input enters your Arduino through the Analog In pins # 0 - # 5. These signals originate from analog sensors and interface devices. These analog sensors and devices use voltage levels to communicate their information instead of a simple yes (HIGH) or no (LOW). For this reason you cannot use a digital pin as an input pin for these devices. Analog Input pins are used only for receiving Analog signals. It is only possible to read the Analog Input pins so there is no command necessary in the `setup( )` function to prepare these pins for input. To read the Analog Input pins use the command:

```
analogRead(pinNumber);
```

where pinNumber is the Analog Input pin number. This function will return an Analog Input reading between 0 and 1023. A reading of zero corresponds to 0 Volts and a reading of 1023 corresponds to 5 Volts. These voltage values are emitted by the analog sensors and interfaces. If you have an Analog Input that could exceed Vcc + .5V you may change the voltage that 1023 corresponds to by using the Aref pin. This pin sets the maximum voltage parameter your Analog Input pins can read. The Aref pin's preset value is 5V.

Digital Input can enter your Arduino through any of the Digital Pins # 0 - # 13. Digital Input signals are either HIGH (On, 5V) or LOW (Off, 0V). Because the Digital pins can be used either as input or output you will need to prepare the Arduino to use these pins as inputs in your `setup( )`function. To do this type the command:

```
pinMode(pinNumber, INPUT);
```

inside the curly brackets of the `setup( )` function where pinNumber is the Digital pin number you wish to declare as an input. You can change the pinMode in the `loop( )`function if you need to switch a pin back and forth between input and output, but it is usually set in the `setup( )`function and left untouched in the `loop( )`function. To read the Digital pins set as inputs use the command:

```
digitalRead(pinNumber);
```

where pinNumber is the Digital Input pin number.

Input can come from many different devices, but each device's signal will be either Analog or Digital, it is up to the user to figure out which kind of input is needed, hook up the hardware and then type the correct code to properly use these signals.

Things to remember about Input:

Input is either Analog or Digital, make sure to use the correct pins depending on type.
To take an Input reading use `analogRead(pinNumber);` (for analog)
Or `digitalRead(pinNumber);` (for digital)
Digital Input needs a pinMode command such as `pinMode(pinNumber, INPUT);`
Analog Input varies from 0 to 1023
Digital Input is always either HIGH or LOW

Examples of Input:

Push Buttons, Potentiometers, Photoresistors, Flex Sensors

SIK Worksheets v.1.0 **Name:**
Output **Date:**

All of the electrical signals that the Arduino works with are either input or output. It is extremely important to understand the difference between these two types of signal and how to manipulate the information these signals represent.

Output Signals

A signal exiting an electrical system, in this case a micro-controller.

Output to the Arduino pins is always Digital, however there are two different types of Digital Output; regular Digital Output and Pulse Width Modulation Output (PWM). Output is only possible with Digital pins # 0 - # 13. The Digital pins are preset as Output pins, so unless the pin was used as an Input in the same sketch, there is no reason to use the pinMode command to set the pin as an Output. Should a situation arise where it is necessary to reset a Digital pin to Output from Input use the command:

```
pinMode(pinNumber, OUTPUT);
```

where pinNumber is the Digital pin number set as Output. To send a Digital Output signal use the command:

```
digitalWrite(pinNumber, value);
```

where pinNumber is the Digital pin that is outputting the signal and value is the signal. When outputting a Digital signal value can be either HIGH (On) or LOW (Off).

Digital Pins # 3, # 5, # 6, # 9, # 10 and #11 have PWM capabilities. This means you can Output the Digital equivalent of an Analog signal using these pins. To Output a PWM signal use the command:

```
analogWrite(pinNumber, value);
```

where pinNumber is a Digital Pin with PWM capabilities and value is a number between 0 (0%) and 255 (100%). For more information on PWM see the PWM worksheets or S.I.K. circuit 12.

Output can be sent to many different devices, but it is up to the user to figure out which kind of Output signal is needed, hook up the hardware and then type the correct code to properly use these signals.

Things to remember about Output:

Output is always Digital
There are two kinds of Output: regular Digital or PWM (Pulse Width Modulation)
To send an Output signal use `analogWrite(pinNumber, value);` (for analog) or `digitalWrite(pinNumber, value);` (for digital)
Output pin mode is set using the pinMode command: `pinMode(pinNumber, OUTPUT);`
Regular Digital Output is always either HIGH or LOW
PWM Output varies from 0 to 255

Examples of Output:

Light Emitted Diodes (LED's), Piezoelectric Speakers, Servo Motors

SIK Worksheets v.1.0 **Activity**
Programming Concepts, Input/Output

Purpose: Group activity teaching the concepts of input and output as used in Arduino Programming and Physical Computing. Text formatted `like this` denotes actual Arduino code.

Materials: Three to five different sized balls and a white/chalk board big enough so the whole room can see it.

Vocabulary to be explained prior to activity:

input: A pin mode that intakes information.
output: A pin mode that sends information.
digitalRead: Command used to get a HIGH or LOW value from a digital input pin.
analogRead: Command used to get a value between or including 0 and 1023 from an analog input pin.
digitalWrite: Command used to send a HIGH or LOW value to an output pin.
analogWrite: Command used to send a PWM value to an output pin simulating an analog output.
PWM: A value between 0 and 255 representing a digital signal simulating an analog output. Used with analogWrite.
HIGH: Electrical signal present (5V for Uno). Also ON or True in boolean logic.
LOW: No electrical signal present (0V). Also OFF or False in boolean logic.

Preparation: Divide the class in quarters, assign each group the following names: Sensors, Input Pins, Output Pins, and Output Components. Arrange the groups in lines in this order about ten to twenty feet apart (or farther if the students are older). The students at the front of each line are Code, their job is to write the Arduino code corresponding to the signal received by their team on the chalk board or white board. Distribute the balls so that each student in the Sensor line has at least one of each size. The smallest balls (tennis or bouncy balls) represent the smallest signal a sensor can send to the Arduino, LOW. The largest balls represent the largest signal a sensor can send to the Arduino, HIGH. The ball or balls of medium size represent PWM values depending on size.

Activity: To start the activity ask the Code student in the Sensor line to tell the class what kind of sensors the Sensor line represents (photoresistor, potentiometer, flex sensor, etc...) and write on the board what the sensor value is. The sensor value corresponds to the sensor type. For example, if the sensor type is photoresistor then “Sunny day” might be written to signal a HIGH signal or “Really cloudy” might be written to signal a smaller PWM value.

Each student in the Sensor line then throws the corresponding sized ball to a student in the Input Pin line. Once all the Input Pins have caught their “signals” the Code student in the Input Pin line writes the analogRead or digitalRead value they think corresponds to the signal.

Once the analogRead value has been written on the board the Input Pin line throws the balls to the Output Pin line. The Code student in that line writes the analogWrite or digitalWrite value they think corresponds to the signal on the board and the balls are thrown to the Output Components.

Once all the Output Components catch their balls the Code student tells the class what type of output component the Output Component line represents and the Output Components strike a pose depending on the signal they received. In the example below poses for LEDs and Servos are shown, but students should be encouraged to make up their own output poses or actions. For example, to represent a HIGH value with motor component outputs students might run in place as fast as they can.

Once the Output Component line has finished, the balls are thrown back to the Sensor line, the Code students are replaced by another student in their line and the process starts over. Once every student has had a chance to be the Code student the lines should switch so eventually everyone has a chance to play each part of the input/output process.

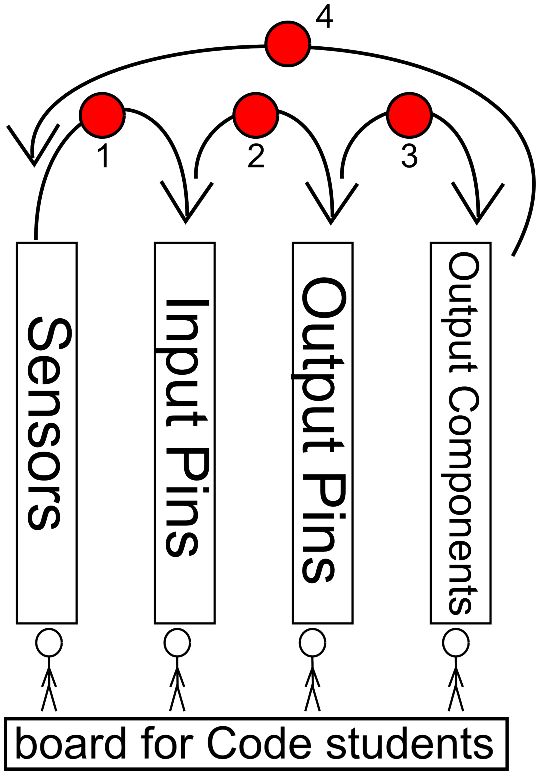
This version of the input/output activity is the simplest form of the activity. If students are comfortable with this version and want more of a challenge there are many ways to complicate the activity.

Give the Output line a set of balls as well as the Sensor line and place a piece of code between the Input and Output lines. The code should be a map command switching the Output signal. For example use:

```
map(signal, 0, 1023, 255, 0);
```

so that the Output line must throw a large ball (HIGH signal) when the Input line receives a small ball (LOW signal). You can then switch this code through out the game.

Get rid of the Code students and have the Sensor line choose which ball they will throw. Each student can yell out what their line's value equals depending on the size of the ball they catch. This version is a little more fun but will also be a little more chaotic.



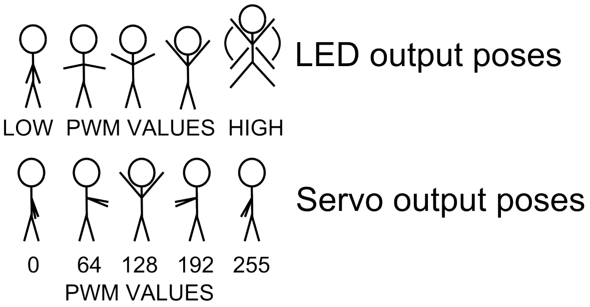
● Tennis ball representing LOW

● Medium-small ball representing PWM value 1

● Medium ball representing PWM value 2

● Medium-large ball representing PWM value 3

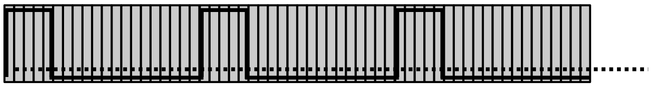
● Large ball representing HIGH

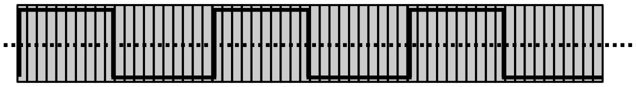


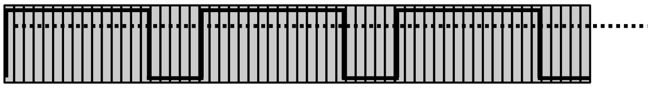
Additional thoughts: This is a great activity just prior to computer lab time. Instead of having kids bouncing off the monitors they will be calmer and ready to sit still applying the concepts they just solidified through physical activity. This is great for kinesthetic learners in particular.

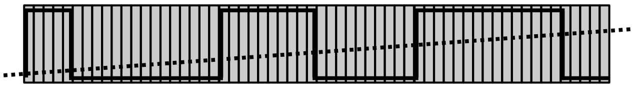
SIK Worksheets v.1.0 **Name:**
PWM **Date:**

For the most part in computer language one means ON and zero means OFF. This keeps things nice and simple, but what if you want to turn something halfway ON so that it is not all the way ON and not all the way OFF? You can't just use a decimal because digital technology only understands ones and zeros. For this reason some of the pins on your Arduino are labeled PWM or Pulse Width Modulation pins. This means you can send a bunch of ones and zeros real quick and the Arduino board will read these ones and zeros as an average somewhere between one and zero. The dotted line in the diagrams represent the average. See tables below.


PWM signal at 25%


PWM signal at 50%


PWM signal at 75%


PWM signal rising from 25% to 75 %

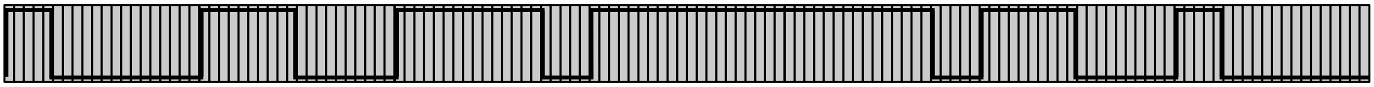
Luckily a lot of the work has been done for you so you don't have to figure out the actual patterns of ones and zeros. All you have to do is pick a number between 0 and 255 and type the command analogWrite. The number zero means the pin is set fully off, the number 255 means the pin is set fully on, and all other numbers set the pin to values between ON (100% or 255) and OFF (0% or 0). You can use PWM on any pin labeled PWM and do not need to set the pin mode before sending an analogWrite command.

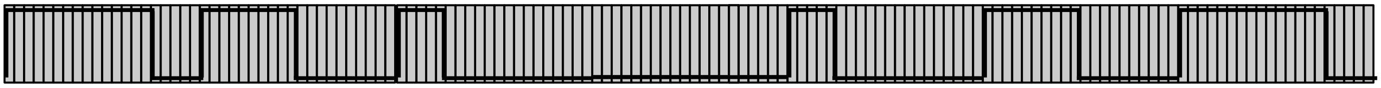
How do you think a PWM signal will affect each of these components compared to a 1 or a 0?
LED : ____

Motor : ____

Piezo : ____

Draw a line through these two charts to show where you believe the PWM value should be.





There are many concepts outside of electrical engineering that are similar to Pulse Width Modulation. Can you list at least three and explain what is being modulated?

____
____
____
____

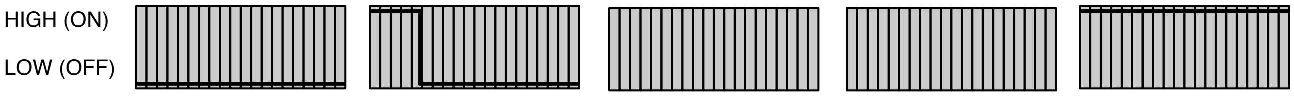
SIK Worksheets v.1.0
PWM

Name:
Date:

Computers and microprocessors only understand two things, ON and OFF. These are represented in a few different ways. There is ON and OFF, One and Zero, or HIGH and LOW. Ones and Zeros are used in the computer language Binary, HIGH and LOW are used with electricity, ON and OFF are plain old human speak.
But what if we want to turn something digital less than 100% ON? Then we use something called PWM, or Pulse Width Modulation. The way your Arduino microprocessor does this is by turning the electricity on a PWM pin ON and then OFF very quickly. The longer the electricity is ON the closer the PWM value is to 100%. This is very useful for controlling a bunch of stuff. For example: the brightness of a light bulb, volume of sound, or the speed of a motor. These are very basic examples, what else might you need to control that is not only ON or OFF? Explain at least two examples.

A microprocessor creates a PWM signal by using a built in clock. The microprocessor measures a certain amount of time (also called a window or a period) and turns the PWM pin ON (or HIGH) for the first part of this window and then OFF (or LOW) near the end of the window. The window is filled up with a different length ON (or HIGH) signal depending on the PWM value. If the PWM value is 50% then the PWM signal is ON (or HIGH) for half of the window. If the PWM value is 25% then the PWM signal is ON (or HIGH) for a quarter of the window. The only time the window will not have a LOW value is if the PWM signal is turned completely ON the whole time and therefore equal to 100% ON. The opposite is true as well, if the PWM signal is set to 0% or OFF, then there will not be any HIGH value at the beginning of the window. Explain in your own words what a PWM window is.

Below are five different PWM windows. A PWM signal is simply a bunch of PWM windows one after another. Some are missing labels and some are missing diagrams. Fill in the blanks.



PWM percent 0% (No wave)

PWM percent __________%

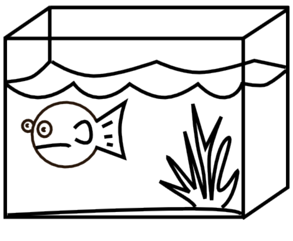
PWM percent 50%

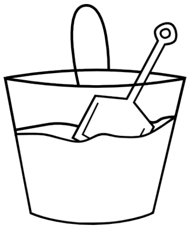
PWM percent 75%

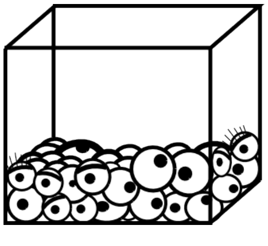
(fill in the window diagrams for these)

PWM percent 100% (No wave)

Below are three different metaphors for a PWM window and a PWM signal. Write the physical item that represents the window and the item or items that represents the signal. Then estimate the PWM percent.







Window: ______________
Signal: __________%___

Window: ______________
Signal: __________%___

Window: ______________
Signal: __________%___

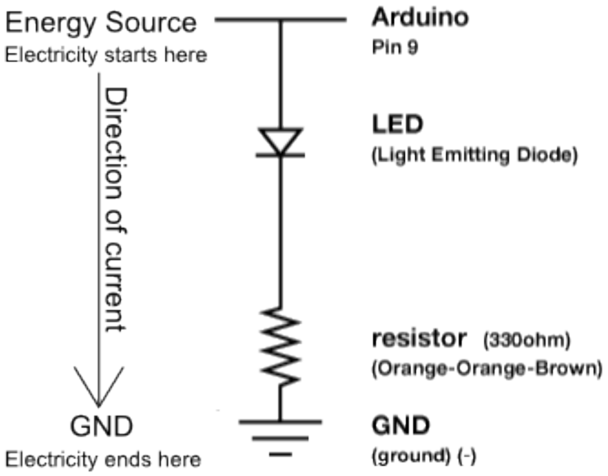
Name:
Date:

Circuit 1

Explanation:

This circuit takes electricity from digital Pin # 9 on the Arduino. Pin # 9 on the Arduino has Pulse Width Modulation capability allowing the user to change the brightness of the LED when using analogWrite. The LED is connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire). The resistor dissipates current so the LED does not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin # 9: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.

LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Also lights up!

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In this circuit the resistor reduces the current so the LED does not burn out.

Gnd: Ground

Code:

```
int ledPin = 3;

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

 void loop() {
   digitalWrite(ledPin, HIGH); //LED on
   delay(1000); // wait second
   digitalWrite(ledPin, LOW); //LED off
   delay(1000); // wait second
}
```

or for PWM output loop could read :

```
int ledPin = 3;

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

 void loop() {
   analogWrite(ledPin, 255); // LED on
   delay(1000); // wait second
   analogWrite(ledPin, 0); // LED off
   delay(1000); // wait second
}
```

This first circuit is the simplest form of output in the kit. You can use the LED to teach both analog and digital output before moving on to more exciting outputs. There is an LED built into your Arduino board which corresponds to Digital Pin # 13.

SIK Worksheets v.1.0
Circuit 2, How the Circuits Work

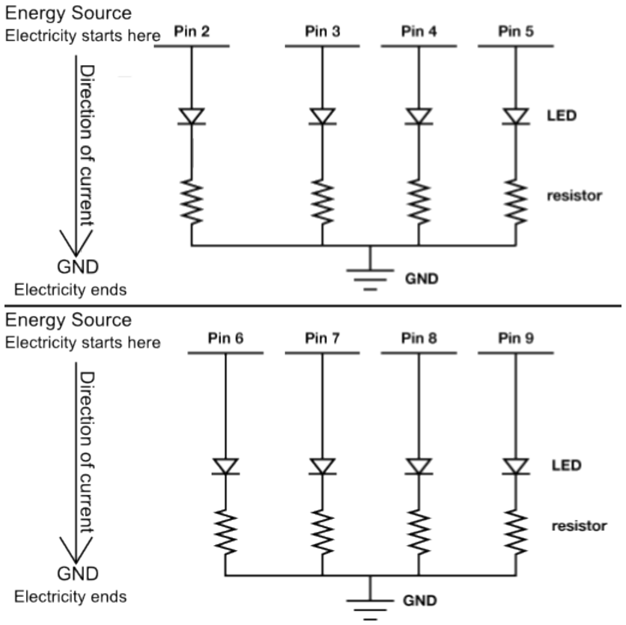
Name:
Date:

Circuit 2

Explanation:

This circuit takes electricity from Pin # 2 through Pin # 9 on the Arduino. The LEDs are connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire). The resistor dissipates current so the LEDs do not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pins # 2 - # 9: Power source, analog (if code uses analogWrite, only possible on pins 3, 5, 6, & 9) or digital (if code uses digitalWrite) output from Arduino board.

LEDs: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Also lights up!

330 Ohm Resistor: The resistors resist the current flowing through the circuit. In this circuit the resistors reduce the current so the LEDs do not burn out.

Gnd: Ground

Code:

```
//this line below declares an array
int ledPins[ ] = {2,3,4,5,6,7,8,9};

void setup( ) {

//these two lines set Digital Pins # 0
- 8 to output
for(int i = 0; i < 8; i++){
pinMode(ledPins[i],OUTPUT);

}

void loop( ) {

//these lines turn the LEDs on and then
off
for(int i = 0; i <= 7; i++){
  digitalWrite(ledPins[i], HIGH);
  delay(delayTime);
  digitalWrite(ledPins[i], LOW);
  }
}
```

The code examples in the S.I.K get a little complicated for the second circuit, but don't worry, it's just more outputs. Some of the code examples use “for” loops to do something a number of times, if you're not familiar with “for” see Loops in Programming Concepts.

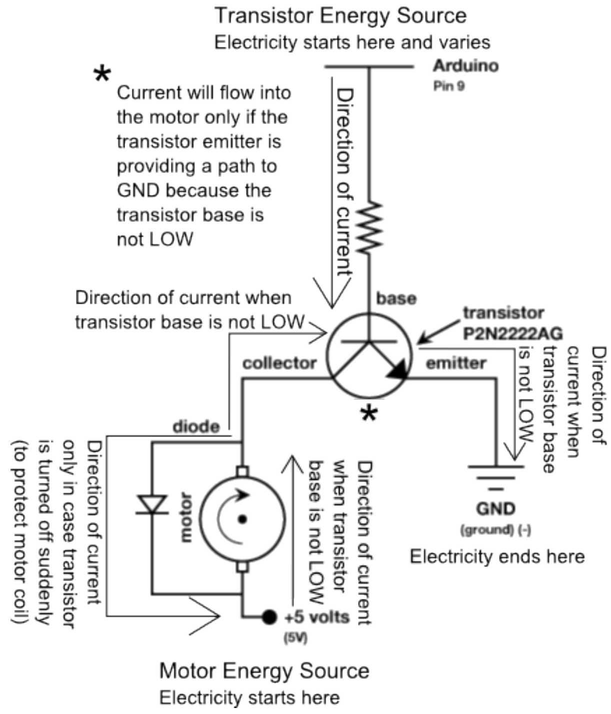
SIK Worksheets v.1.0 **Name:**
Circuit 3, How the Circuits Work **Date:**

Circuit 3

Explanation:

The motor in this circuit takes electricity from 5V on the Arduino. The transistor takes electricity from Pin # 9 on the Arduino. The resistor before the transistor limits the voltage so the PWM output from the Arduino affects the motor rate properly. The higher the voltage supplied to the base of the transistor, the more electricity is allowed through the motor circuit to ground. If the transistor base is LOW no electricity is allowed through to ground and the motor will not run. Pin # 9 on the Arduino has PWM capability so it is possible to run the motor at any percentage. The flyback diode connected close to the motor is simply to protect the motor in the rare case that electricity flows from the transistor towards the motor. This only happens if the transistor is shut off suddenly. Finally, after turning the motor and traveling through the forward biased transistor, the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:


Transistor Energy Source
Electricity starts here and varies
Arduino
Pin 9
*
Current will flow into the motor only if the transistor emitter is providing a path to GND because the transistor base is not LOW
Direction of current
Direction of current when transistor base is not LOW
base
transistor
P2N2222AG
collector
emitter
*
Direction of current when transistor base is not LOW
diode
Direction of current only in case transistor is turned off suddenly (to protect motor coil)
motor
Direction of current when transistor base is not LOW
GND
(ground) (-)
Electricity ends here
+5 volts
(5V)
Motor Energy Source
Electricity starts here

Components:

Arduino Digital Pin # 9: Signal power source, PWM output from Arduino board.

Motor: Electric motor, + and – connections, converts electricity to mechanical energy.

Transistor: A semiconductor which can be used as an amplifier or a switch. In this case the amount of electricity supplied to the base corresponds to the amount of electricity allowed through from the collector to the emitter.

Flyback Diode: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction.

10K Ohm Resistor: A resistor resists the current flowing through the circuit. In this circuit the resistor acts as a 'pull-down' resistor to ground.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
int motorPin = 9;

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


void loop() {
   for (int i = 0; i < 256; i++){
     analogWrite(motorPin, i);
     delay(50);
    }
}
```

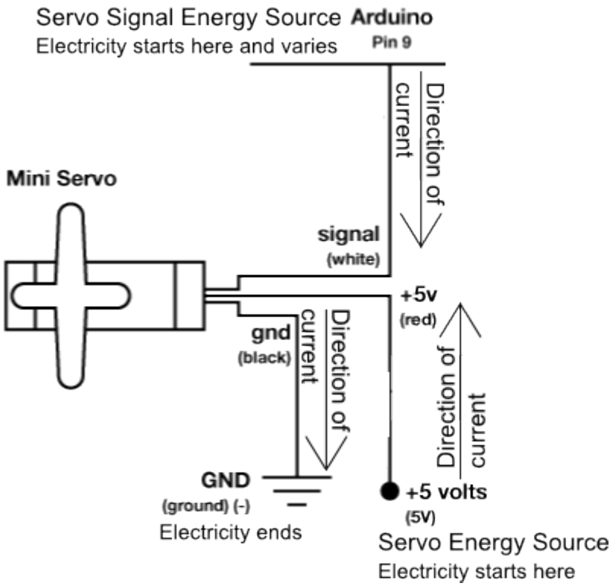
If you are not familiar with electronics that's a lot of information. This circuit is great because it teaches about transistors, one of the basic electronic building blocks.

Circuit 4

Explanation:

The servo in this circuit takes electricity from 5V on the Arduino. Pin # 9 on the Arduino supplies a PWM signal which sets the position of the servo. Each voltage value has a distinct correlating position. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin #9: Signal power source for servo.

Servo: Sets the position of the servo arm depending on the voltage of the signal received.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
//include the servo library for use
#include <Servo.h>
Servo myservo;  //create servo object

int pos = 0;
void setup() {
  myservo.attach(9);
}
void loop() {
//moves servo from 0° to 180°
  for(pos = 0; pos < 180; pos += 1) {
    myservo.write(pos);
    delay(15);
    }
  // moves servo from 180° to 0°
  for(pos = 180; pos>=1; pos-=1)  {
    myservo.write(pos);
    delay(15);
    }
}
```

To some kids this is exciting stuff. There are all kinds of things kids can think to do with servos, you've just got to ask them. Throw out the word “robot” and see what comes back at you. Remember, this is just slightly more complicated output, same as the motor and LED.

SIK Worksheets v.1.0
Circuit 5, How the Circuits Work

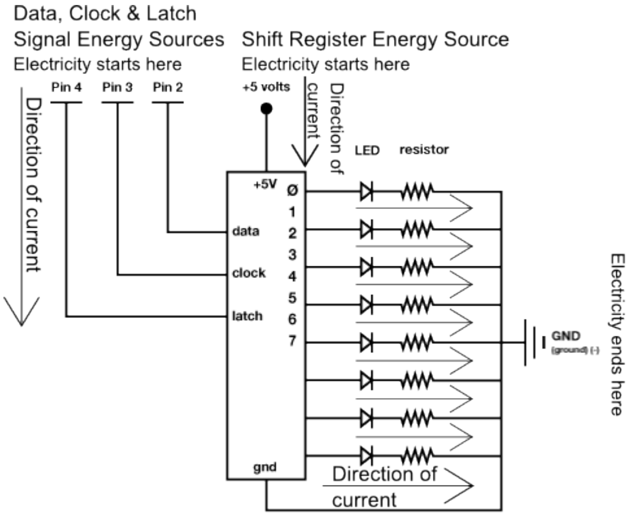
Name:
Date:

Circuit 5

Explanation:

The shift register in this circuit takes electricity from 5V on the Arduino. Pin # 2, # 3 and # 4 on the Arduino supply a digital value. The latch and clock pins are used to allow data into the shift register. The shift register sets the eight output pins to either HIGH or LOW depending on the values sent to it via the data pin. The LEDs are connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire) if the shift register pin is HIGH. The resistor dissipates current so the LEDs do not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin # 2, # 3 and # 4: Signal power source for data, clock and latch pins on shift register.

Shift register: Allows usage of eight output pins with three input pins, a power and a ground. Link to datasheet.

LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Lights up!

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In this circuit the resistor reduces the current so the LED does not burn out.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
int data = 2;
int clock = 3;
int latch = 4;

int ledState = 0;
const int ON = HIGH;
const int OFF = LOW;

void setup() {
  pinMode(data, OUTPUT);
  pinMode(clock, OUTPUT);
  pinMode(latch, OUTPUT);
}

void loop(){
  for(int i = 0; i < 256; i++) {
    updateLEDs(i);
    delay(25);
    }
}

void updateLEDs(int value) {
  digitalWrite(latch, LOW);
  shiftOut(data, clock, MSBFIRST, value);
  digitalWrite(latch, HIGH);
}
```

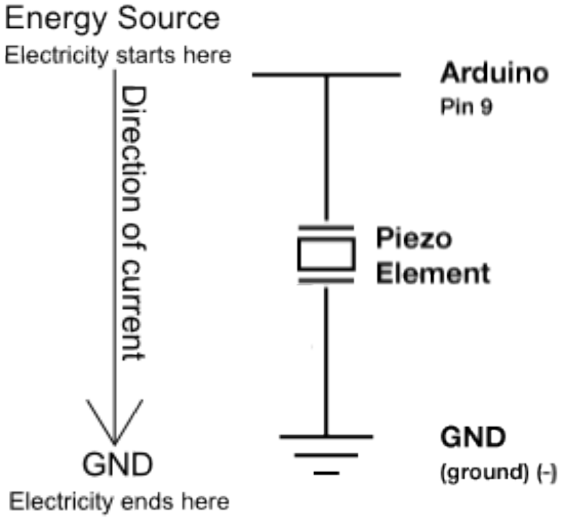
Which brings us to another good point.... For more advanced components you will need to read documentation or Datasheets to figure out how to use them. Any documentation is good as long as you can get the correct information out of it. Datasheets are your friends!

Circuit 6

Explanation:

This circuit gets electricity from Arduino Pin # 9. The Piezo element plays different musical notes depending on the speed and duration of the electrical signal sent from Pin # 9. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin # 9: Power source, digital output from Arduino board. (If changed to PWM output this creates distortion of note, not a change in volume.)

Piezo element: A tiny speaker with a magnetic coil that responds to electrical current by moving more or less depending on the current. The coil is attached to a diaphragm that moves air and causes the noise we hear.

Gnd: Ground

Code:

```
/* this section contains only the two
functions needed to make the piezo play a
note of a given duration. These functions
are called in the loop ( ) function. */

void playTone(int tone, int duration) {
  for (long i = 0; i < duration * 1000L; i
+= tone * 2) {
    digitalWrite(speakerPin, HIGH);
    delayMicroseconds(tone);
    digitalWrite(speakerPin, LOW);
    delayMicroseconds(tone);
   }
}
void playNote(char note, int duration) {
  char names[ ] = { 'c', 'd', 'e', 'f',
'g', 'a', 'b', 'C' };
  int tones[ ] = { 1915, 1700, 1519, 1432,
1275, 1136, 1014, 956 };

  for (int i = 0; i < 8; i++) {
  if (names[i] == note) {
    playTone(tones[i], duration);
      }
   }
}
```

This code is fairly complicated. Don't worry if you don't understand some aspects of it. If you do understand it, congratulations! You are already ahead of this packet in regards to Arduino code. Just remember, this is another way to use digital pins to create analog output.

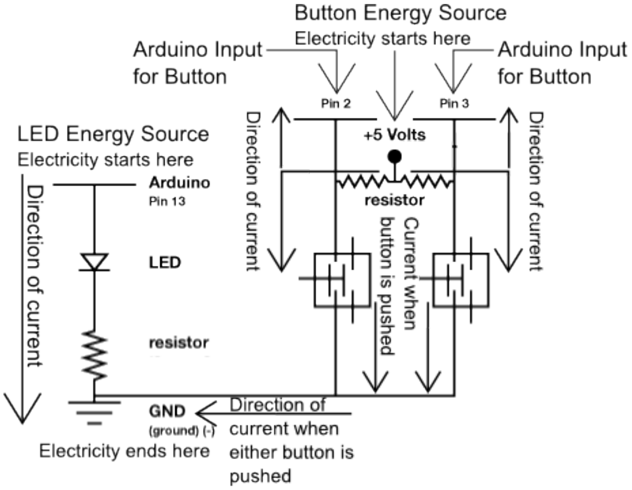
Name:
Date:

Circuit 7

Explanation:

This circuit is actually two different circuits. One circuit for the buttons and another for the LED. See ‘How the Circuits Work’, Circuit 1 for an explanation of the LED circuit. The button circuit gets electricity from the 5V on the Arduino. The electricity passes through a pull up resistor, causing the input on Arduino Pins # 2 and # 3 to read HIGH when the buttons are not being pushed. When a button is pushed it allows the current to flow to ground, causing a LOW reading on the input pin connected to it. This LOW reading is then used in the code you load onto the Arduino and effects the power signal in the LED circuit.

Schematic:



Components:

Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.

Arduino Digital Pin # 2 and # 3: Digital input to Arduino board.

330 & 10K Ohm Resistors: Resistors resist the current flowing through the circuit. In the LED circuit the 330 ohm resistor reduces the current so the LED in the circuit does not burn out. In the button circuits the 10K's ensure that the buttons will read HIGH when they are not pressed.

LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Lights up!

Button: A press button which is open (or disconnected) when not in use and closed (or connected) when pressed. This allows you to complete a circuit when you press a button.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
const int buttonPin = 2;
const int ledPin =  13;

int buttonState = 0;

void setup() {
  pinMode(ledPin, OUTPUT);
  //this line below declares the button
pin as input
  pinMode(buttonPin, INPUT);
}

void loop(){
  //this line assigns whatever the
Digital Pin 2 reads to buttonState
  buttonState = digitalRead(buttonPin);


  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
   }
  else {
    digitalWrite(ledPin, LOW);
   }
}
```

This circuit is the first circuit to use the input capabilities of the Arduino. Notice the difference in `setup( )`. You are still using a Digital Pin but you are using it as input rather than output. Buttons are sweet by the way, let the kids press these buttons instead of yours.

SIK Worksheets v.1.0
Circuit 8, How the Circuits Work

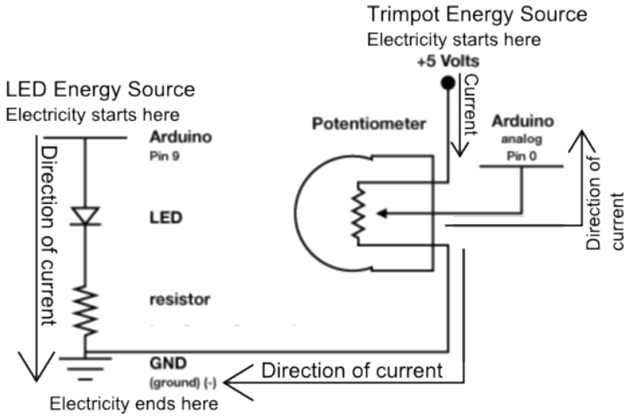
Name:
Date:

Circuit 8

Explanation:

This circuit is actually two different circuits. One circuit for the potentiometer and another for the LED. See How the Circuits Work, Circuit 1 for an explanation of the LED circuit. The potentiometer circuit gets electricity from the 5V on the Arduino. The electricity passes through the potentiometer and sends a signal to Analog Pin # 0 on the Arduino. The value of this signal changes depending on the setting of the dial on the potentiometer. This analog reading is then used in the code you load onto the Arduino and effects the power signal in the LED circuit. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.

Arduino Analog Pin # 0: Analog input to Arduino board.

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In the LED circuit it reduces the current so the LED in the circuit does not burn out.

LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction.

Potentiometer: A voltage divider which outputs an analog value.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
int sensorPin = 0;
int ledPin = 13;
int sensorValue = 0;

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

void loop() {

//this line assigns whatever the
//Analog Pin 0 reads to sensorValue

 sensorValue = analogRead(sensorPin);

 digitalWrite(ledPin, HIGH);
 delay(sensorValue);
 digitalWrite(ledPin, LOW);
 delay(sensorValue);
}
```

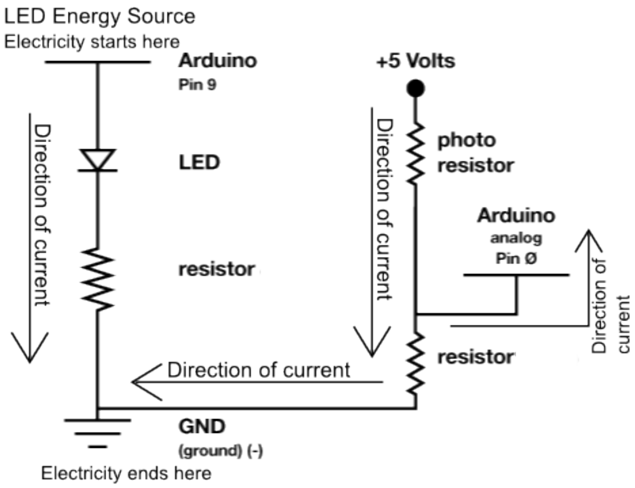
This is another example of input, only this time it is Analog. Circuits 7 and 8 in the S.I.K. introduces you to the two kinds of input your board can receive: Digital and Analog. Not sure what a voltage divider is? Check out the Voltage Divider page towards the back of this section.

Circuit 9

Explanation:

This circuit is actually two different circuits. One circuit for the photoresistor and another for the LED. See How the Circuits Work, Circuit 1 for an explanation of the LED circuit. The photoresistor circuit gets electricity from the 5V on the Arduino. The electricity passes through the photoresistor and sends a signal to Analog Pin # 0 on the Arduino. The value of this signal changes depending on the amount of sunlight. This analog reading is then used in the code you load onto the Arduino and effects the power signal in the LED circuit. The resistor below the Analog Pin connection creates the voltage divider necessary to measure the resistance of the photoresistor. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin # 13: Power source, PWM (if code uses analogWrite) or digital (if code uses digitalWrite) output from Arduino board.

Arduino Analog Pin # 0: Analog input to Arduino board.

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In the LED circuit it reduces the current so the LED in the circuit does not burn out. In the photoresistor circuit the resistor completes the voltage divider.

LED: As in other diodes, current flows easily from the + side, or anode (longer wire), to the - side, or cathode (shorter wire), but not in the reverse direction. Lights up!

Photoresistor: A resistor with a resistance value that changes depending on the amount of light hitting the sensor.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
int lightPin = 0;
int ledPin = 9;

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

void loop() {
  int lightLevel =
analogRead(lightPin);
  lightLevel = map(lightLevel, 0, 900,
0, 255);
  lightLevel = constrain(lightLevel, 0,
255);
  analogWrite(ledPin, lightLevel);
}
```

This circuit is another example of Analog input. It is also a perfect example of a voltage divider. Don't worry about the “map” and “constrain” functions they are explained in the glossary. Unsure about the voltage divider? See the voltage divider page towards the back of this section.

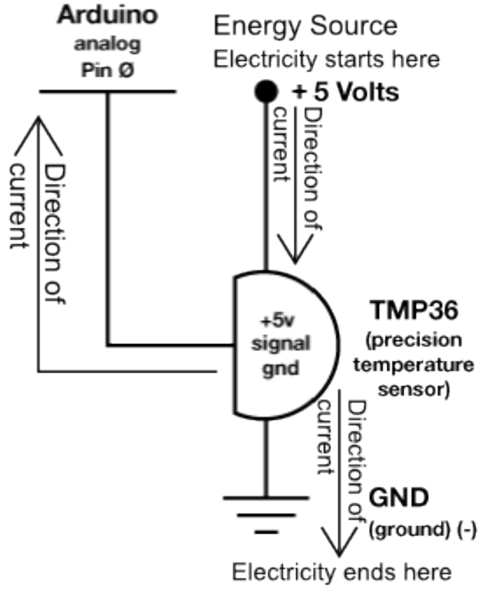
Name:
Date:

Circuit 10

Explanation:

This circuit takes electricity from the 5V on the Arduino. The temp sensor sends an analog value to Arduino Analog Pin # 0. Then the electricity reaches ground, closing the circuit and allowing electricity to flow from power source through the sensor to ground. Finally Arduino uses it's Serial monitor to display the temperature reading.

Schematic:



Components:

Arduino Analog Pin # 0: Analog input to Arduino board.

Temperature Sensor: Provides a voltage value depending on the temperature. Some math is then required to convert this value to Celsius or Fahrenheit.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
int temperaturePin = 0;

void setup() {

//Serial comm. at a Baud Rate of 9600
  Serial.begin(9600);
}

void loop() {

//Calls the function to read the sensor pin
  float temp = getVoltage(temperaturePin);

  //Below is a line that compensates for an
offset (see datasheet)
  temp = (temp - .5) * 100;

  //This line displays the variable
temperature after all the math
  Serial.println(temp);
  delay(1000);
}

//function that reads the Arduino pin and
//starts to convert it to degrees

float getVoltage(int pin) {
  return (analogRead(pin) * .004882814);
}
```

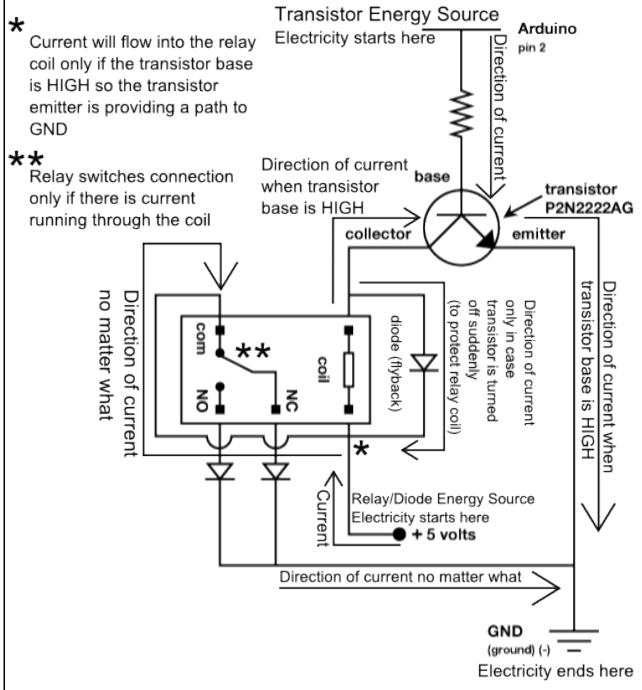
There is a lot of math involved in the code section of this circuit and it all has a reason. But how would you know you need to offset the temperature reading by .5 unless you had read the Datasheet? Also, pay attention to the code lines that enable Serial communication.

Circuit 11

Explanation:

The relay circuit gets electricity from the 5V on the Arduino. The electricity always passes through the relay communication line which is switched to either NO (Normally Open) or NC (Normally Closed), lighting up one of the two LEDs. The transistor gets electricity from Arduino Digital Pin # 2 with a resistor to prevent burn out. In this case the transistor receives a digital signal. The transistor closes the circuit when it is sent a HIGH value, allowing electricity to flow through the relay coil, into the collector, out the emitter and to ground, completing the circuit. The energized coil sets the relay switch to NO. The transistor opens or breaks the circuit when it is sent a LOW value, so no electricity passes through the coil and the relay switch is set to NC. The flyback diode connected close to the motor is simply to protect the motor in the rare case that electricity flows from the transistor towards the motor. This only happens if the transistor is shut off suddenly.

Schematic:


* Current will flow into the relay coil only if the transistor base is HIGH so the transistor emitter is providing a path to GND
** Relay switches connection only if there is current running through the coil
Transistor Energy Source
Electricity starts here
Arduino
pin 2
Direction of current
Direction of current when transistor base is HIGH
base
transistor
P2N2222AG
collector
emitter
Direction of current no matter what
com
**
NO
NC
coil
diode (flyback)
Direction of current only in case transistor is turned off suddenly (to protect relay coil)
Direction of current when transistor base is HIGH
*
Current
Relay/Diode Energy Source
Electricity starts here
+ 5 volts
Direction of current no matter what
GND
(ground) (-)
Electricity ends here

Components:

Arduino Digital Pin # 2: Power source, digital output from Arduino board.
Relay: The relay acts as an electrically operated switch between the two LED's.
Transistor: A semiconductor which can be used as an amplifier or a switch. In this case the amount of electricity supplied to the base corresponds to the amount of electricity allowed through from the collector to the emitter.
330 Ohm & 10K Resistors: A resistor resists the current flowing through the circuit. In the transistor circuit it reduces the current so the transistor in the circuit does not burn out.
Flyback Diode: As in other diodes, current flows easily from the + side, or anode, to the - side, or cathode, but not in the reverse direction. In this case the diode is being used to prevent current from 'flying back' to the relay in case the transistor is suddenly turned off.

Code:

```
int ledPin =  2;

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

void loop() {
  //set the transistor on
  digitalWrite(ledPin, HIGH);
  // wait for a second
  delay(1000);
  // set the transistor off
  digitalWrite(ledPin, LOW);
 // wait for a second
 delay(1000);
}
```

By now you should be thinking that transistors seem pretty important in the world of electrical circuits. They can be used as switches or amplifiers and they are often called the most important invention of the 20th century. The relay is also a great control component, but it needs something to activate it, hence the transistor.

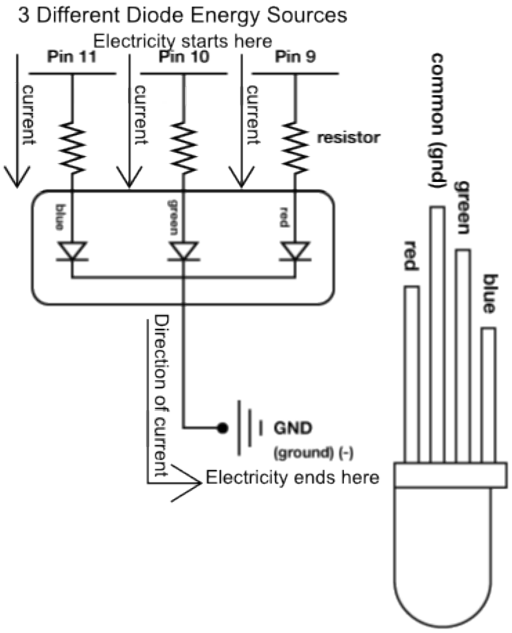
Name:
Date:

Circuit 12

Explanation:

This circuit is pretty straight forward. The Digital Arduino Pins # 9, # 10 and # 11 supply a PWM value to each of the three different LEDs within the Tri-Color LED (Red, Green, and Blue). The LEDs are connected to the circuit so electricity enters through the anode (+, or longer wire) and exits through the cathode (-, or shorter wire). The resistors dissipate current so the LEDs do not draw current above the maximum rating and burn out. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground. By supplying different values to just these three Digital Pins you can mix 16,777,216 different colors!

Schematic:



Components:

Arduino Digital Pin # 9, # 10 and # 11: Power source, PWM output from Arduino board.

RGB LED: Unlike single color LED's, on RGB (Also called 'Tri-Color') LED's, the cathode (or ground wire) is the longest wire and each color (Red, Green, and Blue) gets its own lead. (See the schematic for details).

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In this circuit the resistor reduces the current so the LEDs do not burn out.

Gnd: Ground

Code:

```
const int RED_LED_PIN = 9;
const int GREEN_LED_PIN = 10;
const int BLUE_LED_PIN = 11;
int redIntensity = 0;
int greenIntensity = 0;
int blueIntensity = 0;
const int DISPLAY_TIME = 100;

void setup() {
// No setup required but you still need it
}

void loop(){
  for (greenIntensity = 0; greenIntensity <= 255;
greenIntensity+=5) {
      redIntensity = 255-greenIntensity;
      analogWrite(GREEN_LED_PIN, greenIntensity);
      analogWrite(RED_LED_PIN, redIntensity);
      delay(DISPLAY_TIME);
  }
  for (blueIntensity = 0; blueIntensity <= 255;
blueIntensity+=5) {
      greenIntensity = 255-blueIntensity;
      analogWrite(BLUE_LED_PIN, blueIntensity);
      analogWrite(GREEN_LED_PIN, greenIntensity);
      delay(DISPLAY_TIME);
  }
  for (redIntensity = 0; redIntensity <= 255;
redIntensity+=5) {
      blueIntensity = 255-redIntensity;
      analogWrite(RED_LED_PIN, redIntensity);
      analogWrite(BLUE_LED_PIN, blueIntensity);
      delay(DISPLAY_TIME);
  }
}
```

See how combining just three simple outputs can create some amazing results?

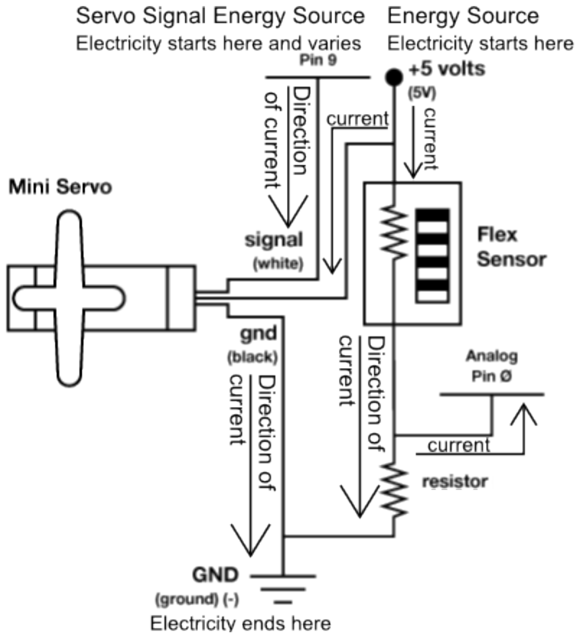
Name:
Date:

Circuit 13

Explanation:

This circuit is actually two different circuits. One circuit for the flex sensor and another for the servo. See How the Circuits Work, Circuit 4 for an explanation of the servo circuit. The flex sensor circuit gets electricity from the 5V on the Arduino. The electricity passes through the flex sensor and sends a signal to Analog Pin # 0 on the Arduino. The value of this signal changes depending on the amount of bend in the flex sensor. This analog reading is then used in the code you load onto the Arduino and sets the position of the servo. The resistor and flex sensor create a voltage divider which is measured by Analog Pin # 0. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pin # 9: Power source, PWM output from Arduino board.

Arduino Analog Pin # 0: Analog input to Arduino board.

10K Ohm Resistor: A resistor resists the current flowing through the circuit.

Flex Sensor: A resistor with a value that varies depending on the amount of bend in the sensor.

Servo: Sets the position of the servo arm depending on the voltage of the signal received.

+5V: Five Volt power source.

Gnd: Ground

Code:

```
#include <Servo.h>
Servo myservo;

int potpin = 0;
int val;

void setup() {
  Serial.begin(9600);
  myservo.attach(9);
}

void loop() {
  val = analogRead(potpin);
  Serial.println(val);
  val = map(val, 50, 300, 0, 179);
  myservo.write(val);
  delay(15);
}
```

This analog input is definitely very different from any other input we have looked at so far but the concept is the same. We treat the sensor as a resistor in a voltage divider to get a reading and then change our output depending on that reading.

SIK Worksheets v.1.0
Circuit 14, How the Circuits Work

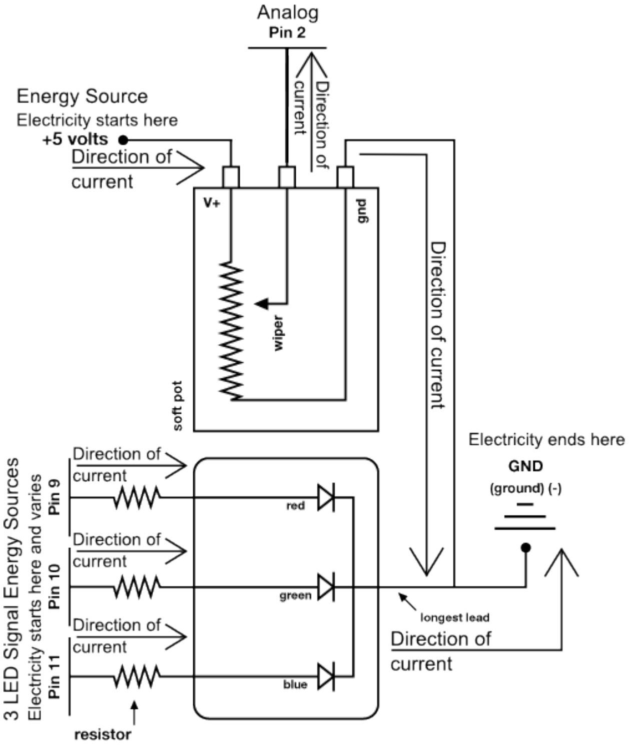
Name:
Date:

Circuit 14

Explanation:

This circuit is actually two different circuits. One circuit for the soft pot and another for the RGB LED. See How the Circuits Work, Circuit 12 for an explanation of the RGB LED circuit. The soft pot circuit gets electricity from the 5V on the Arduino. The electricity passes through the soft pot and sends a signal out the com line of the soft pot to Analog Pin # 0 on the Arduino. The value of this signal changes depending on where the wiper (any type of contact) touches the soft pot. This analog reading is then used in the code you load onto the Arduino and sets the color of the RGB LED. Notice that yet again our sensor and the input pin form a voltage divider, only this time the voltage divider is completely inside the sensor. The wiper divides the resistor into two different portions with values that depend on the position of the wiper. Finally the electricity reaches ground, closing the circuit and allowing electricity to flow from power source to ground.

Schematic:



Components:

Arduino Digital Pins # 9, 10, 11: Power source, PWM output from Arduino board.

Arduino Analog Pin # 0: Analog input to Arduino board.

330 Ohm Resistor: A resistor resists the current flowing through the circuit. In the RGB LED circuit it reduces the current so the LED it is attached to does not burn out.

Flex Sensor: A resistor with a value that varies depending on the amount of bend in the sensor.

RGB LED: A grouping of three LEDs, Red, Green and Blue. Power goes in three different anodes (+, the short wires) and out one common cathode (-, the long wire). Lights up!

+5V: Five Volt power source.

Gnd: Ground

Code:

```
const int RED_LED_PIN = 9;
const int GREEN_LED_PIN = 10;
const int BLUE_LED_PIN = 11;

void setup() {
//No setup necessary but you still need it
}

void loop() {
  int sensorValue = analogRead(0);
  int redValue = constrain(map(sensorValue, 
0, 512, 255, 0),0,255);
  int greenValue = 
constrain(map(sensorValue, 0, 512, 0, 
255),0,255)-constrain(map(sensorValue, 512, 
1023, 0, 255),0,255);
  int blueValue = 
constrain(map(sensorValue, 512, 1023, 0, 
255),0,255);

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

This analog sensor is similar to the flex sensor. You will often see the basic concepts covered in the S.I.K. in different forms as you work with more complicated sensors and outputs. Most of these technologies are built using the same building blocks and some math.

SIK Worksheets v.1.0 **Name:**
Using a Multimeter **Date:**

Often you will have to use a multimeter for troubleshooting a circuit, testing components, materials or the occasional worksheet. This section will cover how to use a digital multimeter, specifically a SparkFun VC830L. We will discuss how to use this multimeter to measure voltage, current, resistance and continuity on the circuits in the S.I.K..

Parts of a multimeter

Display: Where values are displayed.

Knob/Setting: Used to select what is being measured and the upper limit of how much is being measured.

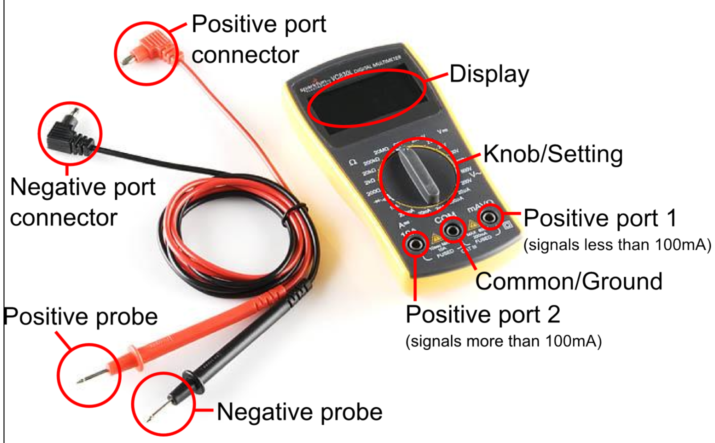
Positive port 1: Where the positive port connector is plugged in if you are measuring less than 100mA of current.

Common/Ground: Where the negative port connector is plugged in no matter what.

Positive port 2: Where the positive port connector is plugged in if you are measuring more than 100mA of current.

Probes: The points of contact for measuring electrical signals. Place the positive probe closer to the energy source and the negative probe closer to ground.

Port connectors: Plug 'em into multimeter.



Important: Sometimes the reading will not remain steady or will display a value that you believe is wrong. If this happens make sure your probes are making firm, constant contact with your circuit on a conductive material.

Settings

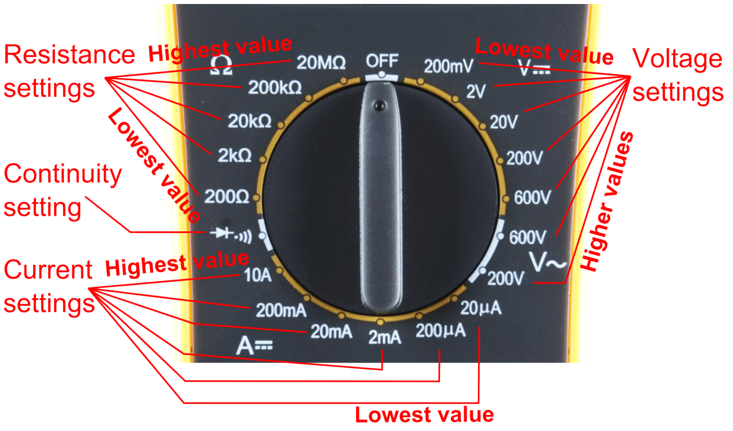
There are a bunch of different settings depending on how much of a signal the multimeter is being used to measure. This is a good opportunity to talk about unit conversion.

Voltage: The options for measuring voltage range from 200mV all the way up to 600 Volts.

Current: The options for measuring current range from 20μA all the way up to 10 Amps.

Resistance: The options for measuring resistance range from 200Ω to 20MΩ.

Continuity: This option is for testing to see if there is an electrical connection between two points.



Changing com ports:

Use the first positive com port if you are measuring a signal with less than 100mA of current. Switch to the second positive com port if you are using more. With the Arduino you will usually be using the first positive com port.

Replacing fuses:

If you try to measure more than 100mA of current through the first positive com port you will most likely blow the fuse in your multimeter. Don't worry, the multimeter isn't broken, it simply needs a new fuse. Replacing fuses is easy, this tutorial explains it: http://www.sparkfun.com/tutorials/202

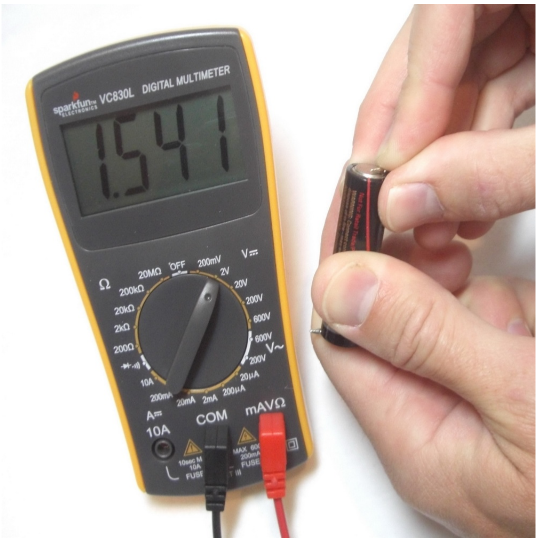
SIK Worksheets v.1.0 **Name:**

Using a Multimeter, Voltage **Date:**

Measuring voltage

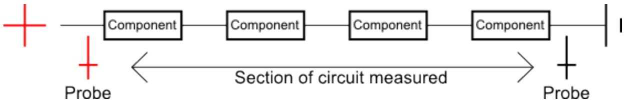
To start with something simple, let's measure voltage on a AA battery: Pull out your multimeter and plug the black probe into COM ('common') jack and the red probe into mAVΩ. Set the multimeter to "2V". Squeeze the probes with a little pressure against the positive and negative terminals of the AA battery. The black probe is customarily connected to ground or '-' and red goes to power or '+'. If you've got a fresh battery, you should see around 1.5V on the display!

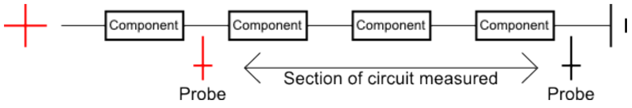
What happens if you switch the red and black probes? Nothing bad happens! The reading on the multimeter is simply negative - so don't worry too much about getting the red or black probe in the right place.

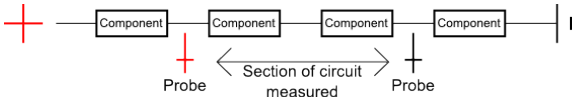


For most Arduino uses you will be measuring voltages that are 9V or less. Knowing this allows you to start your voltage measurement setting at 20V and work your way down.

On a circuit use the multimeter to measure voltage from one point in the circuit to another point somewhere along the same circuit. The multimeter can be used to measure the voltage of the whole circuit (if it's going from 5V to GND this will usually read 4.8 to 5V) or just a portion. If you want to measure the voltage of just a portion of your circuit, you have to pay attention to where you place your probes. Find the portion of the circuit you want to measure, and place one probe on the edge of that portion nearest to the energy source. Place the other probe on the edge of that portion nearest to ground. Voila - you have found the voltage of just that section between your probes! Confused? See the schematic images below. Still confused?For more on this see voltage drop.







If your multimeter reads 1. the multimeter voltage setting you are using is too low. Try a larger voltage setting, if you still encounter the same problem try an even higher setting.

If your multimeter reads 0 the multimeter voltage setting you are using is too high. Try a smaller voltage setting, if you still encounter the same problem try an even smaller setting.

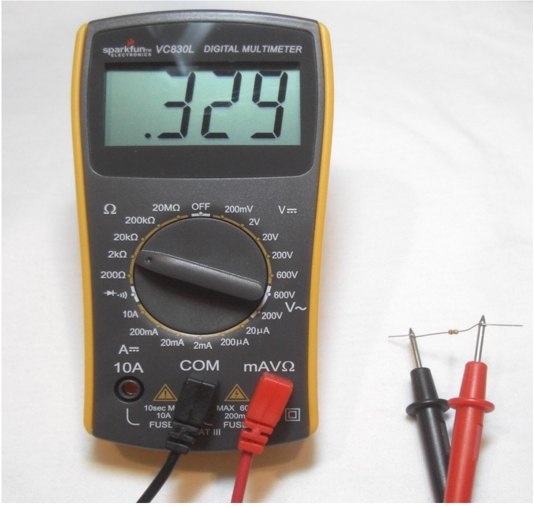
SIK Worksheets v.1.0 **Name:**
Using a Multimeter, Resistance **Date:**

Measuring resistance

To start with something simple, let's measure the resistance of a resistor: Pull out your multimeter and plug the black probe into COM ('common') jack and the red probe into mAVΩ. Set the multimeter to "2kΩ". Squeeze the probes with a little pressure against the wires on either end of the resistor. The black probe is customarily connected to ground or '-' and red goes to power or '+'. The multimeter will measure the resistance of all the components between the two probes.

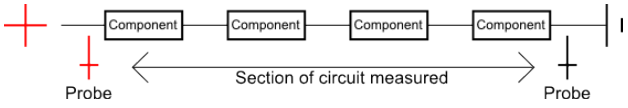
It is important to remember to turn off the power of a circuit before measuring resistance. Measuring resistance is one of the few times you will use a multimeter on a circuit with no power.

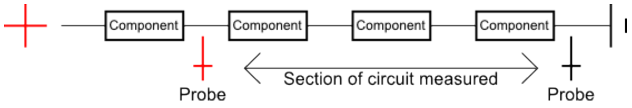
The example to the right is a 330Ω resistor. Notice the multimeter does not read exactly .330, often there is some margin of error.

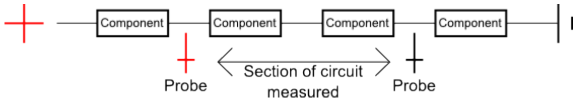


When measuring resistance first make sure that the circuit or component(s) you are measuring do not have any electricity running through them.

On a circuit use the multimeter to measure resistance from one point in the circuit to another point somewhere along the same circuit. The multimeter can be used to measure the resistance of the whole circuit or just a portion. If you want to measure the resistance of just a portion of your circuit, you have to pay attention to where you place your probes. Find the portion of the circuit you want to measure, and place one probe on the edge of that portion nearest to the energy source. Place the other probe on the edge of that portion nearest to ground. Voila - you have found the resistance of just that section between your probes! Confused? See the schematic images below. Still confused? For more on this see resistance.







If your multimeter reads 1. the multimeter resistance setting you are using is too low. Try a larger resistance setting, if you still encounter the same problem try an even higher setting.

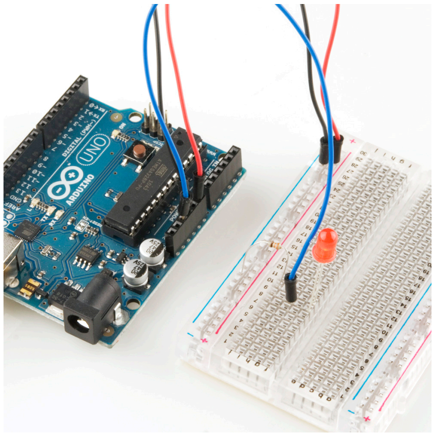
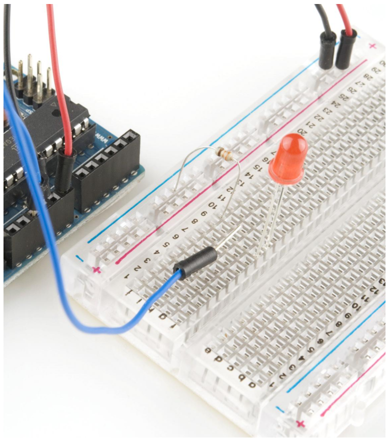
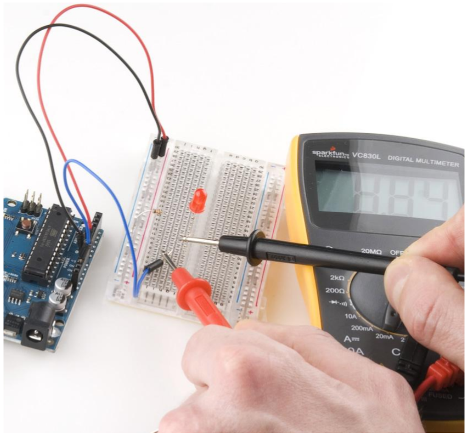
If your multimeter reads 0 the multimeter resistance setting you are using is too high. Try a smaller resistance setting, if you still encounter the same problem try an even smaller setting.

You can measure the resistance of any conductive material whether it is in a circuit or not. Depending on how conductive the material is you may need to change your resistance multimeter setting, or even use a multimeter with a larger range, but if the material is conductive you can measure the resistance of it. This is an easy way to get students to wander around getting comfortable with measuring resistance. Maybe start them off measuring the resistance of some of the S.I.K. circuits, then move to a penny and finally just set them loose to measure anything and everything.

SIK Worksheets v.1.0 **Name:**
Using a Multimeter, Current **Date:**

Measuring current

Ok, we're done with simple. Measuring current is a little more complicated than measuring voltage or resistance. In order to measure current you will need to “break” your circuit and insert the multimeter in series as if the multimeter and it's two probes were a wire. The pictures below are an example of how to measure the current of the first S.I.K. circuit.

		
Unbroken circuit	Circuit broken by unplugging wire connected to power	Multimeter probes touching wire connected to power and positive lead of LED, putting multimeter in series

It doesn't matter where in the circuit you insert your multimeter. The important thing is that the electricity has no choice but to travel through your multimeter in order to get through the rest of the circuit.

So, pull out your multimeter and plug the black probe into COM ('common') jack and the red probe into mAVΩ. Set the multimeter to "20mA". Squeeze the probes with a little pressure against the two wires you used to “break” your circuit. The black probe is customarily connected closer to ground or '-' and red goes closer to power or '+'. The multimeter will measure the total current running through the circuit.

When you are measuring current the multimeter measures the current that is present at that very instant. If your circuit or Arduino is changing the amount of current you will see that change happen instantly on your multimeter. In order to get a good reading make sure you keep the multimeter connected for at least a couple seconds. (You may also get two readings, a high and a low.)

If your multimeter reads 1. the multimeter resistance setting you are using is too low. Try a larger resistance setting, if you still encounter the same problem try an even higher setting.

If your multimeter reads 0 the multimeter resistance setting you are using is too high. Try a smaller resistance setting, if you still encounter the same problem try an even smaller setting.

SIK Worksheets v.1.0
Using a Multimeter, Continuity

Name:
Date:

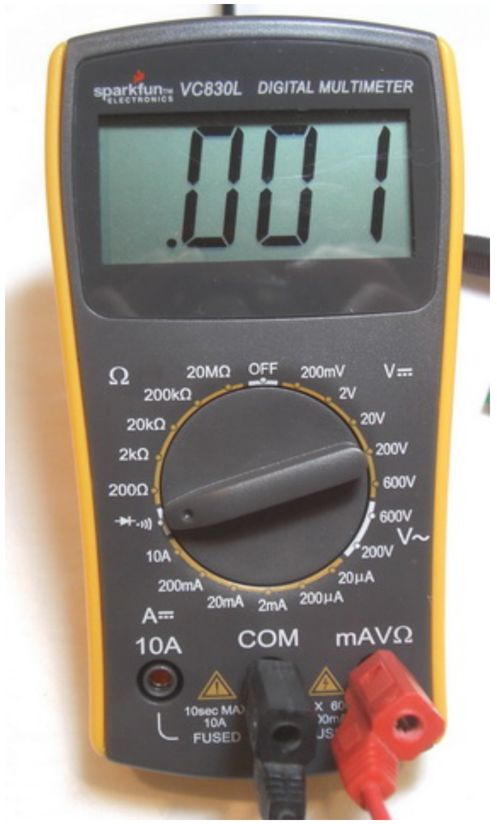
Measuring Continuity

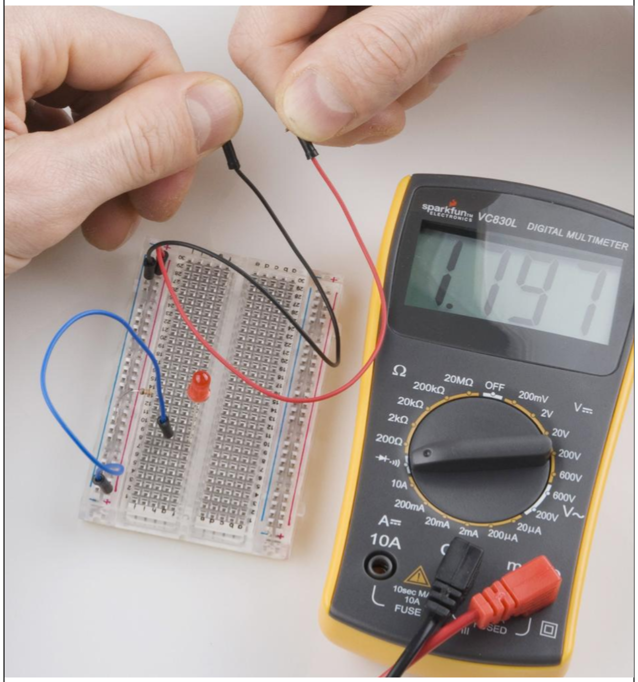
Continuity is how you check to see if two pieces of a circuit are actually connected. The multimeter does this by sending a very small current from the positive probe to the negative, when there is electricity present the multimeter beeps. This is useful when you have a circuit that you think should work but doesn't. Make sure to turn power off when checking continuity.

Set the multimeter to the continuity setting as shown to the right. Touch your probes together and you should hear a beep. This means that electricity is free to travel between the two probes without too much resistance.

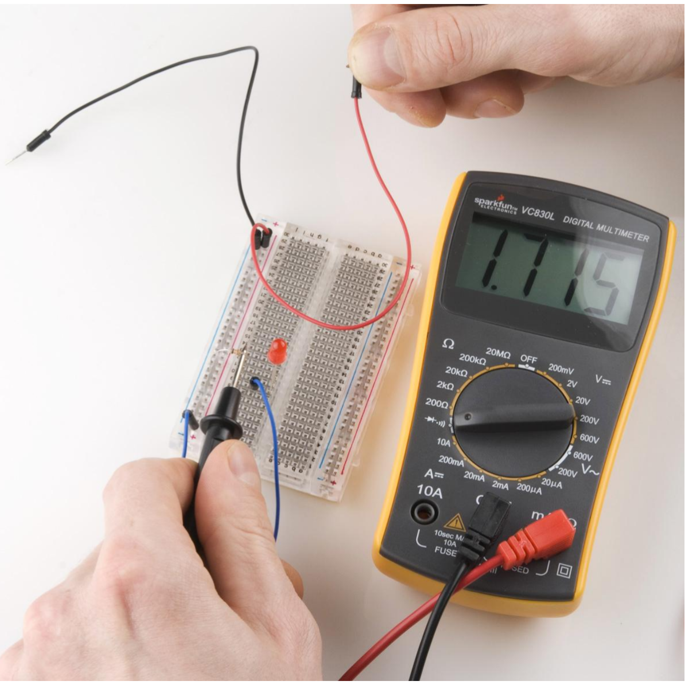
If your circuit is plugged in incorrectly, or if it is broken somewhere (maybe your breadboard or a wire is broken) when you touch the probes to the wire providing power and the wire connected to ground the multimeter will not beep. If it were hooked up correctly you would hear a beep and you wouldn't be using the continuity setting!

In order to figure out where the circuit is broken move one of the probes along the circuit towards the other probe. Do this component by component. When the multimeter beeps you know that the probes now detect electricity passing between them so the break must be between where the probe you are moving is now, and where it was the last time the multimeter didn't beep.





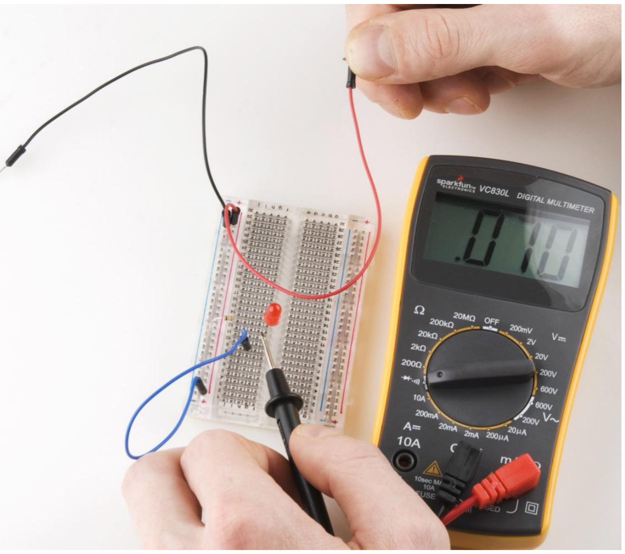
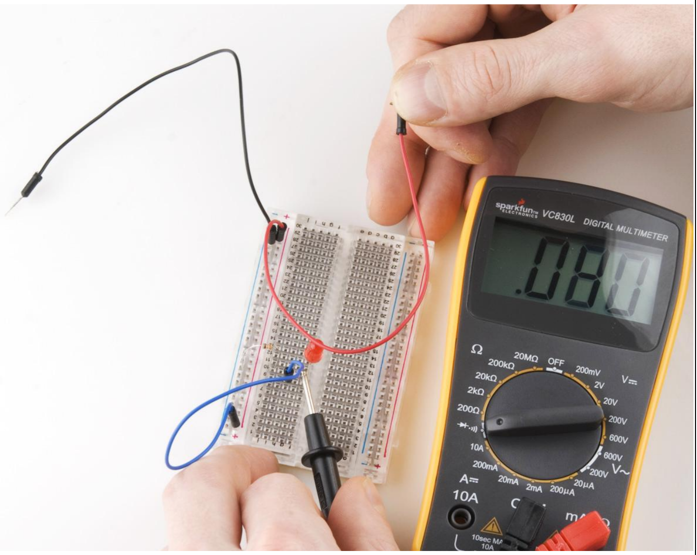
Measuring continuity of the whole circuit from power to ground. Probes are touching wires normally plugged into power and ground.



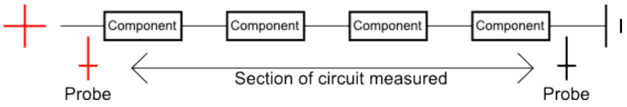
Measuring continuity of the circuit excluding wire connected to ground and resistor. Measured continuity includes LED and two wires connected to breadboard power rail and power. Probes are touching resistor wire and power.

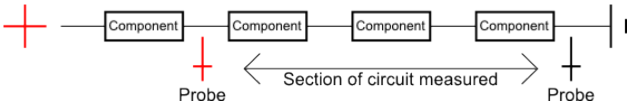
Name:
Date:

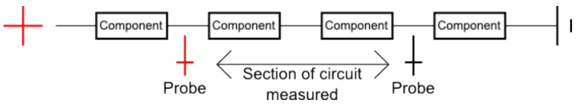
Measuring Continuity II

	
Measuring continuity of the circuit excluding wire connected to ground, resistor and LED. Measured continuity includes two wires connected to breadboard power rail and power. Probes are touching negative LED wire and power.	Measuring continuity of the circuit excluding wire connected to ground, resistor and LED. Measured continuity includes two wires connected to breadboard power rail and power. This image looks similar to the image on the right, but the black probe is touching the blue wire, not the negative LED wire. Probes are touching blue wire and power.

The multimeter can be used to measure the continuity of the whole circuit or just a portion. If you want to measure the continuity of just a portion of your circuit, you have to pay attention to where you place your probes. Find the portion of the circuit you want to measure, and place one probe on the edge of that portion nearest to the energy source. Place the other probe on the edge of that portion nearest to ground. Voila - you are testing the continuity of just that section between your probes! Confused? See the schematic images below.







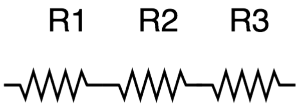
Continuity is one of the most useful settings on a multimeter and you will most likely use it constantly simply to check for connections that aren't quite connected. Breadboards sometimes break so if your multimeter tells you there is no continuity but you know everything is plugged in correctly try switching breadboards.

The beep of the multimeter only tells you that there is very little resistance between the two probes. If there is a resistor in the circuit you will not hear a beep but the display will show a number indicating there is continuity between the two probes.

Series and Parallel

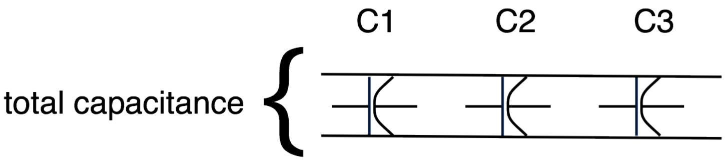
One of the most important concepts in circuit building is the difference between components in **series** and components in **parallel**. Basically you can think of components in series as being one *after* another, like in a chain, while parallel components as hooked up *next* to each other. It's important to know how certain components affect your circuit when hooked up in these two ways. There are a few things to remember, mostly that resistors and inductors work in the opposite way from capacitors:

Resistors and Inductors **in series** can simply be added together:



R1 + R2 + R3 = total resistance

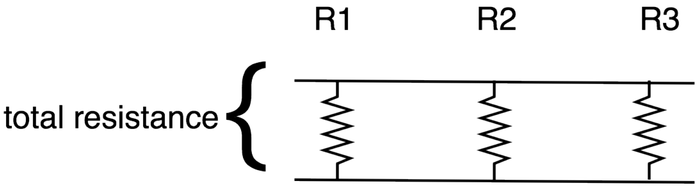
As can capacitors that are **in parallel**:



C1 + C2 + C3 = total capacitance

However, for resistors and inductors **in parallel,** as well as capacitors *in series*, the equation is a bit more complex. Basically the values between any two elements in these setups equal the product of the values divided by the sum of the values. For three elements or more, solve for two and repeat until done. For example (the // indicates that the elements are in parallel):

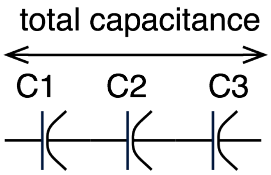
Resistors:



R1 // R2 = (R1 * R2) / (R1 + R2)

Total Resistance = (R1// R2) * R3 / (R1//R2) + R3

Capacitors:



C1 // C2 = (C1 * C2) / (C1 + C2)

Total Capacitance = (C1// C2) * C3 / (C1//C2) + C3

It seems quite dry, but you never know when basic knowledge like this will come in handy. (Hint: read the next section on powering your projects).

Powering Your Projects

When dealing with electronics, it is always a good idea to know how much power you need and how you're going to get it. If you want your project to be portable, or run separately from a computer, you'll need an alternate power source. Plus, not all Arduino projects can be run off 5V from the USB port. Fortunately there are a lot of options, one or more of which should suit your purposes perfectly.

Understanding Battery Ratings

One popular way to get power to your project is through batteries. There are tons of different kinds of batteries (AA, AAA, C, D, Coin Cell, Lithium Polymer, etc.). In fact, there are too many to go over here, however, they all have a few things in common which can help you choose which ones to use. Each battery has a positive (+) and negative terminal (-) that you can think of as your power and ground. Batteries also have ratings in volts and milliamp hours (written mAh). Given this info along with how much current your circuit will draw, you can figure out how long a battery will last. For example, if I have a battery rated at 1.2v for 2500 mAh, and my circuit requires 100mA (milliamps) current, my battery will last around 17.5 hours. Wait, what? Why not 25 hours you say? Well, you shouldn't drain your battery completely, and other factors such as temperature and humidity can affect battery life, so typically the equation for determining battery life is:

(Capacity rating of battery (in mAh) ÷ Current Consumption of Circuit) x 0.7

Note that we could still use our 2500 mAh battery in a 500mA circuit, but then our battery life would only be 3.5 hours. Make sense? There's a lot to understand about powering circuits, so don't worry if it's not all clicking. Just take an educated guess, be safe, use your multimeter, and make adjustments.

It is also worth mentioning that batteries are not the only potential source of power for your project. If your project will be outside or near a window, consider using solar power. There's plenty of good documentation online, but basically, solar cells have the same kinds of voltage and current ratings that any power source might have; the only difference is that the percentage you get from your solar panel depends on how much sunlight it's getting. Check out http://www.solarbotics.com/ for some good products and documentation using solar power.

Powering Your Projects

So, what if your circuit needs 12v, and all you have are a bunch of 1.5v batteries? Or what if you need your project to be powered for longer, but you don’t want to give it too much power? This is where your knowledge of series and parallel may actually come in handy.

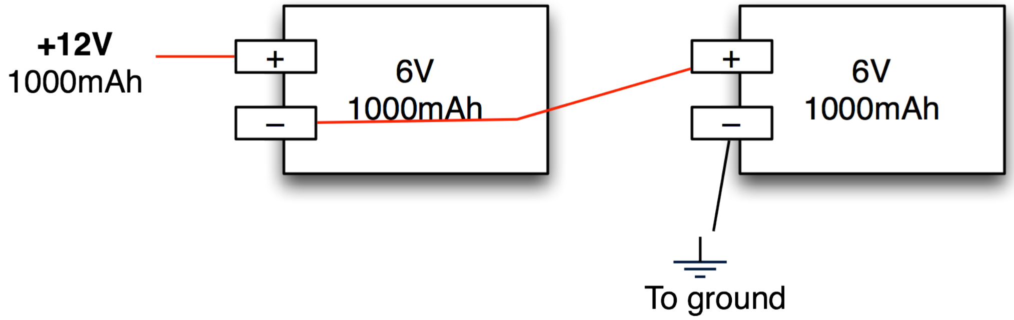
Here’s the rule:

Connecting batteries **in series** increases the voltage but maintains the capacity (mAh) - this what you want to do if you need more power.

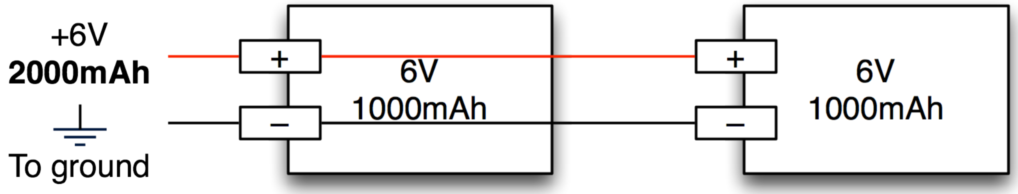
Connecting batteries **in parallel** maintains the voltage but increases the capacity. This is what you want to do if you need your power supply to last longer.

Here’s how to hook them up:

Batteries in Series



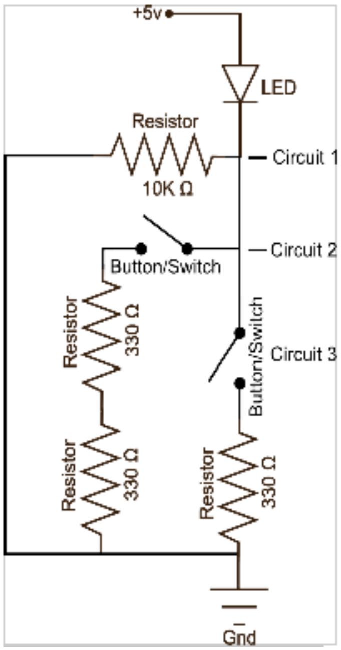
Batteries in Parallel



As always, use caution. Batteries of the same kind (same voltage and capacitance) work best in these kinds of situations. Using different kinds of batteries may also work but it is not recommended, as the results are not as predictable.

SIK Worksheets v.1.0 **Name:**
Resistance **Date:**

Resistance is an important concept when you are creating circuits. Resistance is the difficulty a current encounters when it passes through a component. Everything that electricity passes through provides some measure of resistance, wires, motors, sensors, even the human body!
Measuring voltage, current and resistance are all done in different ways. To measure resistance you disconnect (turn off) your circuit and place both multimeter leads on either side of the portion of the circuit you wish to measure. For example: for measuring just a component you would place your leads on the power and ground leads of the component. To measure the resistance of multiple components you leave them connected and place the positive (red) multimeter lead closer to the disconnected power source and the negative (black) multimeter lead closer to the ground. Sometimes you will want to measure the resistance of input and output leads, but more often you will find yourself measuring resistance along the power to ground circuit. It is important to know how much resistance is present in components and circuits for many reasons. Too much resistance and the current will never travel through the whole circuit, too little and the current may fry some of your components! But most importantly you can use resistance to choose the path the current takes through your circuit.



Hook up the circuit to the right using red LEDs. (Don't hook up the power yet.)

Measure the resistance of each of the possible paths the current can take from power (5v) to ground. There are three possible paths. You will have to measure each component separately and then add the resistance up for the total. You will can add the component's resistance together because the components are in series, if they were parallel it would require more math. Record the total resistance for each circuit below. (Hint: you won't be able to measure the LED)

Circuit 1: _______Ω Circuit 2: _________Ω Circuit 3: ___________Ω
Now connect the power and, one at a time, press the two buttons. Which circuit makes the LED the dimmest? Circuit # ______
If you press both buttons which path does the current take? Circuit # ______
If the voltage is staying at 5v in this circuit no matter which paths are closed, there is a way to calculate the current given the resistance. Write the name of the law and the equation that solves for resistance below. Label all variables.
____

Now measure the resistance of a potentiometer when it is dialed all the way up and down. Record the highest and lowest values you get.

Highest:______________________ Ω

Lowest:_____________________ Ω

Redraw the schematic above (use the back of the worksheet if necessary) but use a potentiometer to control the LED brightness instead of the buttons and various resistors. Remember that you must have at least 330Ω of total resistance, otherwise you'll burn out your LED!

Since a circuit or component does not need a current running through it in order to measure the resistance you can take your multimeter and measure the resistance of anything you can think of. Wander around and measure the resistance of various objects. Start with a penny. Record the most interesting things that have resistance and the value of their resistance below. List at least three.

____

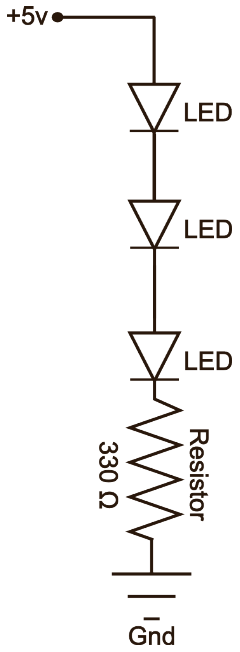
____

____

SIK Worksheets v.1.0 **Name:**
Voltage Drop **Date:**

Voltage drop is an important concept when you are creating circuits. Voltage drop is the amount that the voltage drops when it passes through a component. The following exercises will show how to measure voltage drop in real life. This is essential when you are fixing your remote control car, electric guitar or even a cell phone.
Measuring voltage, current and resistance are all done in different ways. To measure voltage you connect your positive (red) multimeter lead to the side of the circuit that closer to your power source and the negative (black) multimeter lead to the side of the circuit that is closer to the ground. It is important to know how much voltage is going through a circuit for many reasons. The most important reasons being that too much voltage can damage your components and too little voltage may not allow electricity to flow all the way through to ground.

Hook up the circuit to the right using red LEDs.



Close the circuit so only one LED is grounded with the 300Ω resistor. Insert the end of the resistor not plugged into the ground into a hole on the same row as the first LED's negative lead. The other LEDs don't light up, why is this?

____

____

____

Measure the voltage drop across just the LED and record. ________________v

Measure the voltage drop across the LED and the resistor. ________________v

Close the circuit so two LEDs light up.

Voltage drop across one LED = _____v Voltage drop across two LEDs = _____v

Measure the voltage drop across the whole circuit and record. _____________v

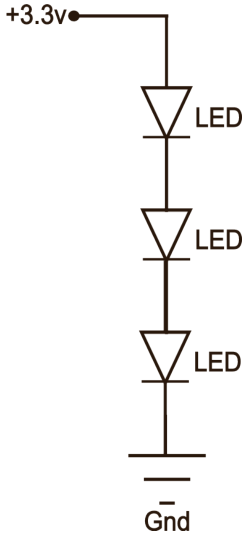
Close the circuit so three LEDs light up.

Voltage drop across one LED = _____v Voltage drop across two LEDs = _____v

Voltage drop for three LEDs = _____v Voltage drop for whole circuit = _____v

What happened to the LEDs with the last question?

____



Now hook up the same circuit to the 3.3V power source without the resistor.
Why do you think you don't need the resistor?

____

____

Measure the voltage drop across all the LEDs and record. ____________v

Close the circuit so only two LEDs light up.

Voltage drop across one LED = _____v Voltage drop across two LEDs = _____v

---

Hook up the circuit above to the 5V power source but use the 3.3v as ground.
Wait a second! You can't use a power source as a ground! Or can you?

What is the voltage available and how many LEDs can you light up with it?

Voltage available = ________v # of LEDs you can light up = __________
Many people think of Gnd as the ONLY place to connect a 'negative' pin, but all you need is a voltage drop from the beginning of a circuit to the end. This difference in voltage is what draws the current in the correct direction.

SIK Worksheets v.1.0 **Name:**
Transistors **Date:**

What is a transistor?

Transistors are semiconductors used to amplify an electrical signal or switch an electrical signal on and off.

Why is a transistor useful?

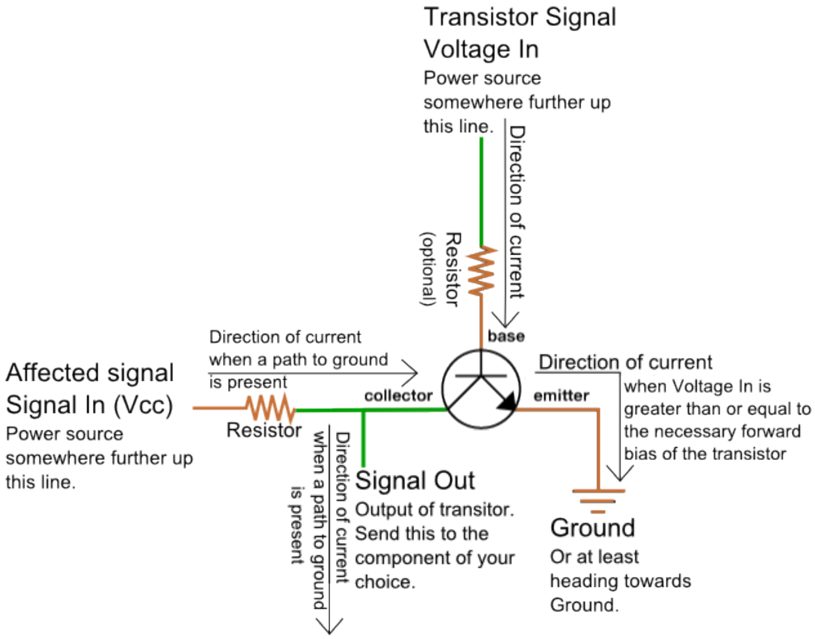
Often you will need more power to run a component than your Arduino can provide. A transistor allows you to control the higher power signal by breaking or closing a circuit to ground. Combining this higher power allows you to amplify the electrical signal in your circuit.

What is in a transistor?

A transistor circuit has four parts; a signal power source (connects to transistor base), an affected power source (connects to transistor collector), voltage out (connects to transistor collector), and ground (connected to transistor emitter).

How do you put together a transistor?

It's really pretty easy. Here is a schematic and explanation detailing how:


Transistor Signal Voltage In
Power source somewhere further up this line.
Direction of current
Resistor (optional)
base
Direction of current when a path to ground is present
Affected signal Signal In (Vcc)
Power source somewhere further up this line.
Resistor
collector
Direction of current when a path to ground is present
Signal Out
Output of transitor. Send this to the component of your choice.
emitter
Direction of current
when Voltage In is greater than or equal to the necessary forward bias of the transistor
Ground
Or at least heading towards Ground.

The transistor voltage in signal is the signal that is used to control the transistor's base.

Signal in is the power source for the signal out which is controlled by the transistor's action.

Signal out is the output of the signal originating from signal in, it is controlled by the collector.

The amount of electrical current allowed through the transistor and out of the emitter to ground is what closes the entire circuit, allowing electrical current to flow through signal out.

Ok, how is this transistor information used?

It depends on what you want to do with it really. There are two different purposes outlined above for the transistor, we will go over both.

If you wish to use the transistor as a switch the signal in and voltage in signal are connected to the same power source with a switch between them. When the switch is moved to the closed position an electrical signal is provided to the transistor base creating forward bias and allowing the electrical signal to travel from the signal in to the transistor's collector to the emitter and finally to ground. When the circuit is completed in this way the signal out is provided with an electrical current from signal in.

The signal amplifier use of the transistor works the same way only Signal In and Voltage In are not connected. This disconnection allows the user to send differing values to the base of the transistor. The closer the voltage in value is to the saturation voltage of the transistor the more electrical current that is allowed through the emitter to ground. By changing the amount of electrical current allowed through to ground you change the signal value of signal out. For examples of transistor uses see S.I.K. circuits # 3 and # 11.

SIK Worksheets v.1.0
Voltage Dividers

Name:
Date:

What is a voltage divider?

Voltage dividers are a way to produce a voltage that is a fraction of the original voltage.

Why is a voltage divider useful?

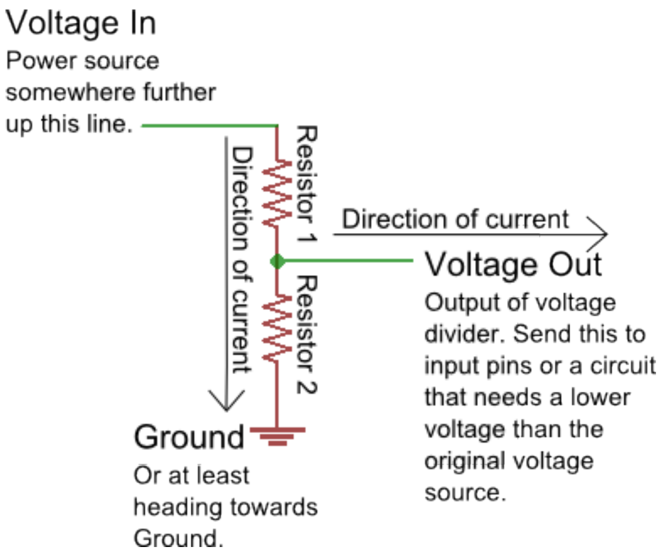
One of the ways a voltage divider is useful is when you want to take readings from a circuit that has a voltage beyond the limits of your input pins. By creating a voltage divider you can be sure that you are getting an accurate reading of a voltage from a circuit. Voltage dividers are also used to provide an analog Reference signal.

What is in a voltage divider?

A voltage divider has three parts; two resistors and a way to read voltage between the two resistors.

How do you put together a voltage divider?

It's really pretty easy. Here is a schematic and explanation detailing how:


Voltage In
Power source somewhere further up this line.
Resistor 1
Direction of current
Direction of current
Voltage Out
Output of voltage divider. Send this to input pins or a circuit that needs a lower voltage than the original voltage source.
Resistor 2
Ground
Or at least heading towards Ground.

Often resistor # 1 is a resistor with a value that changes, possibly a sensor or a potentiometer.

Resistor # 2 has whatever value is needed to create the ratio the user decides is acceptable for the voltage divider output.

The Voltage In and Ground portions are just there to establish which way the electrical current is heading, there can be any number of circuits before and after the voltage divider.

Here is the equation that represents how a voltage divider works:

$$V_{out} = V_{in} \frac{R_2}{(R_1 + R_2)}$$

If both resistors have the same value then Voltage Out is equal to ½ Voltage In.

Ok, how is this voltage divider information used?

It depends on what you want to do with it really. There are two different purposes outlined above for the voltage divider, we will go over both.

If you wish to use the voltage divider as a sensor reading device first you need to know the maximum voltage allowed by the analog inputs you are using to read the signal. On an Arduino this is 5V. So, already we know the maximum value we need for Vout. The Vin is simply the amount of voltage already present on the circuit before it reaches the first resistor. You should be able to find the maximum voltage your sensor outputs by looking on the Datasheet, this is the maximum amount of voltage your sensor will let through given the voltage in of your circuit. Now we have exactly one variable left, the value of the second resistor. Solve for R2 and you will have all the components of your voltage divider figured out! We solve for R1's highest value because a smaller resistor will simply give us a smaller signal which will be readable by our analog inputs.

Powering an analog Reference is exactly the same as reading a sensor except you have to calculate for the Voltage Out value you want to use as the analog Reference.

Given three of these values you can always solve for the missing value using a little algebra, making it pretty easy to put together your own voltage divider. The S.I.K. has many voltage dividers in the example circuits. These include: Circuits # 7, 8, 9, 13 and 14.

Basic Operators and Comments Reference Sheet

Basic Operators

Often when you are programming you will need to do simple (and sometimes not so simple) mathematical operations. The signs used to do this vary from very simple to confusing if you've never seen them before. Below is a table of definitions as well as some examples:

Arithmetic operators	**Relational operators**	**Logical operators**
+ (addition) **-** (subtraction) ***** (multiplication) **/** (division) **%** (modulus) **=** (assignment)	**==** (equality) **!=** (inequality) **>** (greater-than) **<** (less-than) **>=** (greater-than-or-equal-to) **<=** (less-than-or-equal-to)	**!** (NOT) **&&** (AND) **||** (OR)
Arithmetic operators are your standard mathematical signs (no example provided)	Relational operators are used to compare values and variables	Logical operators are used to join two or more conditional statements together

Pay attention to = and ==. = is used to assign variable values, == to compare values.

Relational operator example:

```
if (x!=7){
//loop body code here
}
```

Compares x to the number 7, executes code inside body loop if the value of x **does not equal** 7

Logical operator example:

```
if ((x==7)||(x==9)){
//loop body code here
}
```

Compares x to the number 7 and 9, executes code inside body loop if the value of x equals 7 **or** 9

Comments

As you use code other people have written you will notice //, /* and */ symbols. These are used to “comment” lines out so they do not effect the code. This way people who write code can add comments to help you understand what the code does. Good code has comments that explain what each block of code (functions, classes, etc....) does but does not explain simpler portions of the code as this would be a waste of time. Commenting lines out is also a very useful tool when you are writing code yourself. If you have a section of code you are working on, but isn't quite finished or doesn’t work, you can comment it out so it does not effect the rest of your code when you compile or upload it.

//	**/***	***/**
This is used comment out a single line	This is used to start a section of commented lines	This is used to end or close a section of commented lines
`//commented out line`	`/*comments start here`	`comments end here*/`

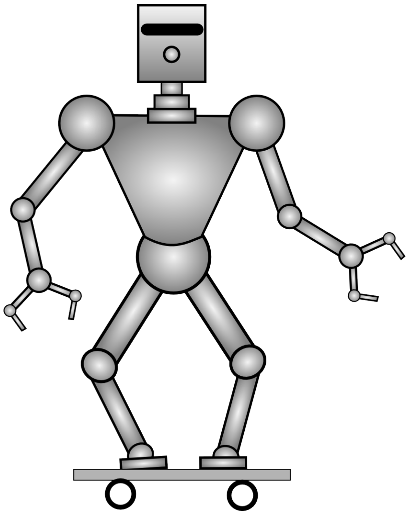
SIK Worksheets v.1.0 **Name:**
Programming Concepts, Variables **Date:**

Vocabulary: Variable, Boolean, Integer, Character, Value

Variables are one of the most important concepts in computer programming. But what exactly are **variables**? **Variables** are like baskets that hold pieces of information. There are a couple different kinds of **variables** depending on what kind of information you need to keep track of. You have probably already heard of most of the different kinds of **variables**. Here are the definitions of three different kinds of **variables**. There are more types of **variables**, but to start with these are the most important types of **variables**.

- **Boolean variable**: A boolean variable can be true or false (one or zero).
- **Integer variable**: An integer variable can be any whole number from –32768 to 32767.
- **Character variable**: A character variable can be any one letter (or punctuation or symbol).

Below is a robot, answer the questions to the right of the robot and be as silly as you want. Then write the type of **variable** you would use to store this information. For a **boolean** write “boolean”, for an **integer** write “int” and for a **character** write “char”.



Is this robot good at skateboarding? ___________ Variable type:_____________
How old is this robot?______ Variable type:_____________
What is the first letter of this robot's name?____________ Variable type:_____________
How many years has it been skateboarding?___________ Variable type:_____________
Is it wearing pants?________ Variable type:_____________
What is the first letter of the robot's dog's name? _______ Variable type:_____________
Is the robot going to crash?__ Variable type:_____________
How many feet of air has this robot gotten? ____________ Variable type:_____________

The number, or character, you put into a **variable** is called its **value**. Once you have created a **variable** you can change the **value** whenever you need to. For example, if we decided the robot is 1000 years old, in a year we need to be able to change its age to 1001. First we need to create a **variable** to keep track of its age. We can name the **variable** whatever we want, but “age” makes sense so we'll go with that. Then we need to put a **value** into the **variable**. The first **value** was 1000, but a year later we delete that **value** and replace it with the new **value**, 1001. Pretty easy, huh? If we wanted to keep track of how old the robot used to be when we met it we could create a new **variable** called “ageWeMet”. That way when we have to change the “age” **variable** we can keep track of how old the robot was when we met it in the other **variable** “ageWeMet”. You may have noticed that there are no spaces in the name of this second **variable**. That is because **variable** names can't have any spaces.

Circle the **variable** in the sentences below and put a box around the **value**.

The robot's favorite letter is Q. The robot's height is 100 ft. The robot's power is on.

SIK Worksheets v.1.0 **Name:**
Programming Concepts, Boolean Variables **Date:**

Vocabulary: Boolean, Declare, Assign

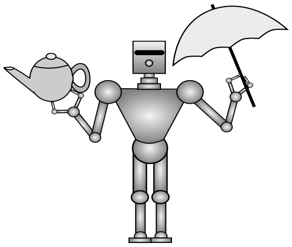
OK! You're ready to start programming your first **boolean** variable. Anytime you see `italics like this` it is an example of how you would write something in the Arduino computer language.

- A **Boolean** variable is the simplest kind of variable, it is either true or false.
- True is represented by a one or HIGH and false is represented by a zero or LOW.
- HIGH can be used as true, but it means there is electricity flowing through a circuit.
- LOW can be used as false, but it means there is no electricity flowing through a circuit.
- To create a **Boolean** variable you type the following: `boolean variableName;`
- Creating a variable is called "**declaring**" a variable.
- The variableName can be anything you like, but it should make sense to you.

For example you could **declare** a **Boolean** variable named `dayLight(boolean dayLight;)` that represents whether it is daytime or not. Once you have **declared** your variable it is not equal to anything, it is empty and waiting for you to set it equal to true or false. To do this you type the following: `dayLight = true;` or `dayLight = 1;`. (Don't forget the ; at the end, it's very important! It is called a semicolon and it tells the computer that you are done doing something.)

This means that dayLight is true, and you can see the sun. Setting a variable equal to a value is called "**assigning**". **Declare** three **Boolean** variables about the robot on this page in the spaces below and then **assign** them values of true or false (or one or zero). Remember, you can name the variables whatever you want! They're your variables, it's up to you. Look at the example above if you are unsure of how to **declare** and **assign**. (Don't forget the semicolons at the end of each line, they're important!)

Declare:			
Assign:			



List three of the silliest things you can think of that you might keep track of with a **boolean** variable. Examples: Do I have peanut butter in my ear? Are penguins good to use as dodgeballs?

Now pick one of the silly ideas above. In the space below **declare** your silly variable and then **assign** it a value. For example: `boolean peanutButter; peanutButter = true;` This means that I do have peanut butter in my ear... maybe I am saving it for lunch.

SIK Worksheets v.1.0 **Name:**
Programming Concepts, Integer Variables **Date:**

Vocabulary: Integer, Declare, Assign

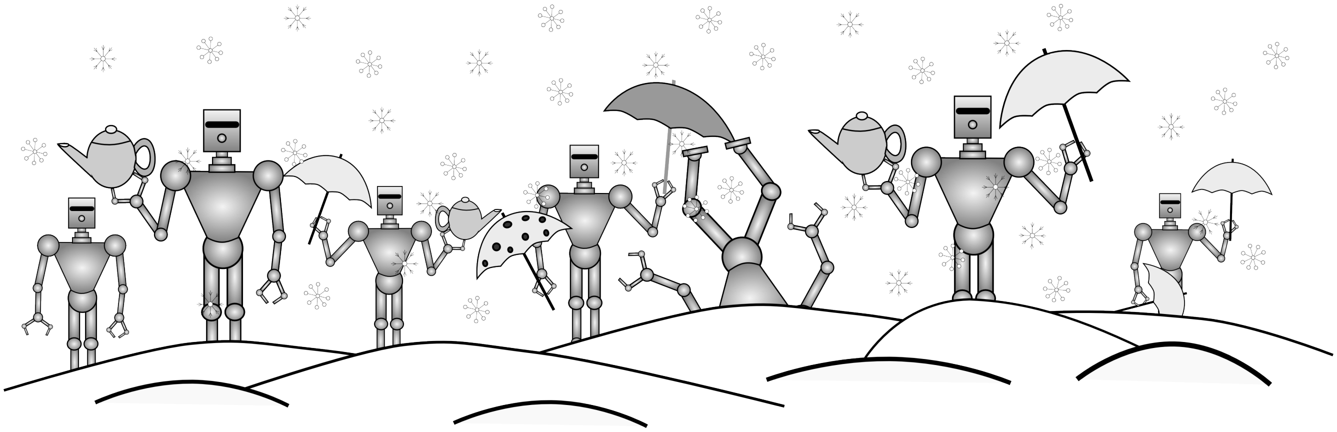
Wow! You're ready to start programming your first **integer** variable. Anytime you see `italics` it is an example of how you would write something in Arduino computer language.

- An **Integer** variable is a number (no fractions or decimals) between -32768 and 32767.
- To create a **Integer** variable you type the following: `int variableName;`
- This is called "**declaring**" a variable.
- The variableName can be anything you like, but it should make sense to you.
- To **assign** an **Integer** variable the value 120 type the following: `variableName = 120;`

For example you could **declare** an **Integer** variable named clouds (`int clouds;`) that represents the number of clouds in the sky. Once you have **declared** your variable it is not equal to anything, it is empty and waiting for you to set it equal to a number between –32768 and 32767. To do this you type the following: *clouds = 8;*. (Don't forget the ; at the end. This is called a semicolon and it's how the computer knows you are done doing something.)

This means that you can see eight clouds in the sky. Setting a variable equal to a value is called "**assigning**". **Declare** three **Integer** variables about the picture on this page in the spaces below and then **assign** them values between –32768 and 32767. Include at least one variable with a negative value and one variable with a value greater than ten. Feel free to make up variables and values that you can't actual see in the picture, just try to keep it sort of making sense. Look at the example above if you are unsure of how to **declare** and **assign**. (Don't forget the semicolons at the end of each line!)

Declare:			
Assign:			



List three of the silliest things you can think of that you might keep track of with an **integer** variable. Example: How many pieces of ham do I have in my pocket? How many bugs could you fit in a rocket?

Now pick one of the ideas above. In the space below declare your variable and **assign** it a value. For example: `int ham; ham = 1073;` Either I have big pockets or small pieces of ham.

SIK Worksheets v.1.0 **Name:**
Programming Concepts, Char Variables **Date:**

Vocabulary: Character, Declare, Assign

OK! You're ready to start programming your first **character** variable. Anytime you see *italics* it is an example of how you would write something in the Arduino computer language.

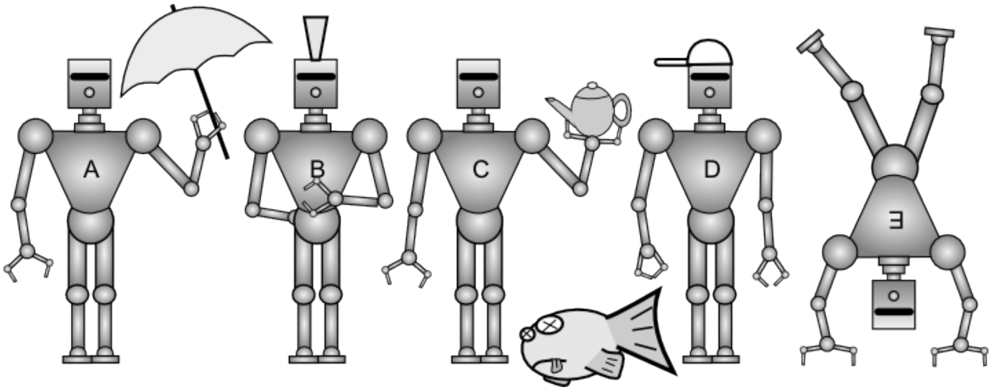
- A **Character** variable is a single letter, symbol or number.
- To create a **Character** variable you type the following: `char variableName;`
- This is called “**declaring**” a variable.
- The variableName can be anything you like, but it should make sense to you.
- To **assign** a **Character** variable the value “Q” you type the following:
  `variableName = 'Q';`

For example you can **declare** a **character** variable named weather (`char weather;`) that uses a letter to represents the weather. You can use the letter R to mean it is raining, S for snow, and C for clear. Once you have **declared** your variable it is not equal to anything, it is empty and waiting for you to set it equal to a character.

To do this you type the following: `weather = 'C';`. (Don't forget the ; at the end. This is called a semicolon and it's how the computer knows you are done doing something.) Also, there are many different **character** types other than a letter: !?*%$&@ are all valid characters.

For example, `weather = 'C';` means that the sky is clear, but that's just because you decided it means that. C could mean whatever you need to keep track of. For example C could mean that it is cold out, if that's what you decided. Setting a variable equal to a value is called “**assigning**”. **Declare** three **Character** variables about the picture on this page in the spaces below and then **assign** them **character** values that make sense. Check the example when your are **assigning** a value, this can get tricky. Make sure the variable names describe the object you want to keep track of. Look at the example above if you are unsure of how to **declare** and **assign**. (Don't forget the quotation marks and semicolons at the end of each line!)

Declare:			
Assign:			



List three of the silliest things you can think of that you might keep track of with a **Character** variable. Example: What color lollipops do robots eat? What’s a pirate’s favorite letter?

Programming Concepts, Variables

Purpose: Group activity teaching how to declare and assign the variable types Boolean, Integer and Character. Text in italics denotes actual Arduino code.

Materials: None.

Vocabulary to be explained prior to activity:

Variable: A way to store a piece of information that may change.
Value: Piece of information assigned to a variable.
Declaration: Creating a variable, when you declare a variable it has no value.
Assignment: Sets or resets the value of a variable.

Types of variables:

- ☐ **Boolean:** This variable type has only two values. True or false, which can also be represented as one and zero or HIGH and LOW. Arduino syntax: `boolean`
- ☐ **Integer:** This variable type is used to store whole numbers. Because Arduino uses two bytes to store integers it can only store numbers from –32768 to 32767. Arduino syntax: `int`
- ☐ **Character:** This variable is used to store any character you can type on a keyboard (and some you can't). It is basically an integer, but it is used for letters and characters. It is mainly used to print messages or send messages when human interaction is needed. Arduino syntax: `char`

Declaring variables:

Boolean: *`boolean variableName;`*

- ☐ variableName can be anything as long as it makes sense and has no spaces in it.
- ☐ Example: *`boolean pamHappy;`* This variable could be used to indicate if Pam is happy or not. Remember the semicolon, it's important!

Integer: *`int variableName;`*

- ☐ variableName can be anything as long as it makes sense and has no spaces in it.
- ☐ Example: *`int pamAge;`* This variable could be used to indicate how old Pam is. Remember the semicolon, it's important!

Character: *`char variableName;`*

- ☐ variableName can be anything as long as it makes sense and has no spaces in it.
- ☐ Example: *`char pamShirtColor;`* This variable could be used to indicate the color of Pam's shirt. Remember the semicolon, it's important!

Assigning variables:

Assigning variables is really easy! No matter what type of variable you simply type the variable name followed by a single equals sign and then the value you are assigning to your variable followed by a semicolon. Example: *`pamShirtColor = 'p';`* Values have certain requirements depending on their types. A boolean needs to be true or false (or one or zero), an integer should be a number between -32768 and 32767 and a character should be a single character with single quotation marks around it. Finally, ***remember the semicolon, it's important!***

Activity:

Students should have completed the introduction to variables worksheet that comes with this activity. Examples of variable types, declarations and assignments can be posted somewhere visible in the classroom to help students who are not completely comfortable with the concepts yet.

Students go around in a circle declaring variables that apply to themselves and other students. For example, if they wish to declare a variable about their age they would need to declare an integer variable with a name that makes sense. It is up to the students how specific they want to get, they can declare an integer variable named age, or they could go so far as to declare a variable named pamAge. The difference is that the variable age can apply to anyone, the variable pamAge is specific to a person named Pam. A boolean variable can be used for any quality that is either yes or no. For example, a student might declare pamHappy as a boolean variable to indicate whether Pam is happy or not. Character variables can be used to keep track of anything that does not fit nicely into either integer or boolean. For example, a student may create a variable called pamShirtColor. Declaration of variables should be in the syntax used in Arduino, for examples see previous page.

Once each student has declared a variable go around the circle and have each student assign a value to their variable. Assignment of variables should be in the syntax used in Arduino, for examples see previous page.

Additional activities:

Students can declare their variables on pieces of construction paper. Each variable type should have a distinct color or shape (or some other way to identify the variable type other than the declaration). Students can write their variable declaration and assignment for display and personalize the construction paper so it makes sense with their variable name. Throughout the unit students should be encouraged to reassign the value assigned to their variable if it changes. Obviously you will probably want to have a designated time for variable reassignment to avoid classroom disruption. For example, Pam may declare *`char pamShirtColor;`* on a shirt shaped piece of yellow construction paper (yellow to designate it a character variable). Pam can then tape a piece of paper with the letter 'B' (don't forget the single quotation marks) to indicate she is wearing a blue shirt. The next day Pam may then replace the letter 'B' with a 'P' to indicate that today she is wearing a purple shirt. You may want to limit reassignment to once a week if your class has a tendency to be overzealous about activities like this.

If your students are having difficulty with the concept of variable types try this activity. Create three different shaped holes in a board, designate one hole for each of the three variable types. Label each hole with the corresponding variable type and definition. Create or buy a bunch of objects that can only fit through one of the holes and label the objects with values that correspond to the variable type. Give the objects out to students and explain that each object can only be one of the three different type of variables and the students need to match up the objects with the variable types by putting them in the corresponding holes.

SIK Worksheets v.1.0 **Name:**
Programming Concepts, If statements **Date:**

Vocabulary: If, parenthesis, curly brackets

The If statement is one of the most basic building blocks in computer programming. The easiest way to understand a computer language If statement is to look at real life If statements first. If statements have two different parts, the question and what happens if the answer to the question is yes. Below are a bunch of real life if statements. On the left are the questions or "if" portions of the If statements. On the right are the actions that happen when the answer to the questions are true. Unfortunately only the first If statement is connected to the correct action, the rest are up to you.

Draw a line between the two that make the most sense together.

Question	**Action**
If you play around with electronics -------------	Then you can build some cool stuff.
If you run over a porcupine with your bike	Then your feet will smell funny.
If you are an alien	Then you pollute less.
If you do push ups and pull ups	Then you say Arrrrr a lot.
If you put peanut butter in your sock	Then you have feathers and don't like cats.
If you eat too much candy	Then you might catch a fish or fall in.
If you bike everywhere you go	Then you might have six arms and one eye.
If you go fishing in a canoe	Then someone might sing Happy Birthday.
If you are a pirate	Then you get stronger.
If you today is your Birthday	Then you get a flat tire.
If you are a parakeet	Then you get sick.

In computer programming the If statement works the same way as real life. There is a question and something that happens if the answer to the question is "yes". The question is written inside of the parenthesis ( ) and whatever happens if the question is true is written inside of the curly brackets { }.

Here are a couple examples of pseudo-code versions of If statements:
If (you play around with electronics){then you can build some cool stuff}
If (you remember parenthesis and curly brackets){then If statements are easy}
If (you understand If statements){then you are on your way to learning programming}
Just remember: If (the answer to this question is yes) {then do this}
Example of an If statement in Arduino:

```
if ( val == HIGH ) {
digitalWrite ( ledPin, LOW );
}
```

All If statements start with "`if`" followed by the question in parenthesis. In this example the question is does the variable "`val`" equal HIGH? (HIGH is a boolean value that is the same as true. HIGH means there is electricity present and LOW means there is not.) If "`val`" does equal HIGH then Arduino does whatever is inside of the two curly brackets { }. In this case it tells `ledPin` it should not conduct electricity. Here is a pseudo-code of the same If statement:

If (the variable "val" has electricity running through it) {then tell (the pin ledPin, to turn off) }

If parts of this last example don't make sense don't worry, the important thing is to understand what an If statement is. So... If (the last example didn't make sense) {don't worry.}

SIK Worksheets v.1.0 **Name:**
Programming Concepts, If statements **Date:**

Write three of the silliest, or most interesting, If statements you can think of in the space below, don't worry about putting them inside of parenthesis and curly brackets, we'll get to that later.

Example 1: If dinosaurs were still alive then we would have to run a lot more.

____

____

Now write your If statements the way they would look with the parenthesis. Don't forget the difference between the two different kinds of parenthesis!

Example 1: If (dinosaurs were still alive) {then we would have to run a lot more.}

____

____

But what if there are two or more things that could happen if the question is true?

Example 2: If dinosaurs were still alive then we would have to run when we were outside, but if they were our pets we could walk and we would need really big litter boxes.

Is this really just one If statement? No, it's actually two, and one of the If statements is inside the other. Don't worry! This is ok, in fact it happens all the time. Here is how it looks in pseudo-code:

```
Example 2:    If (dinosaurs were still alive){
                      then we would have to run a lot more, but
                      If (they were our pets) {
                              we could walk and we would need really big litter boxes} }
```

It may look complicated but it's just one If statement inside of another. There is no limit to how many Ifs you can put inside of another If statement. Go ahead and write one If statement with another If statement inside of it in plain English below. Make sure you use the word “if” twice.

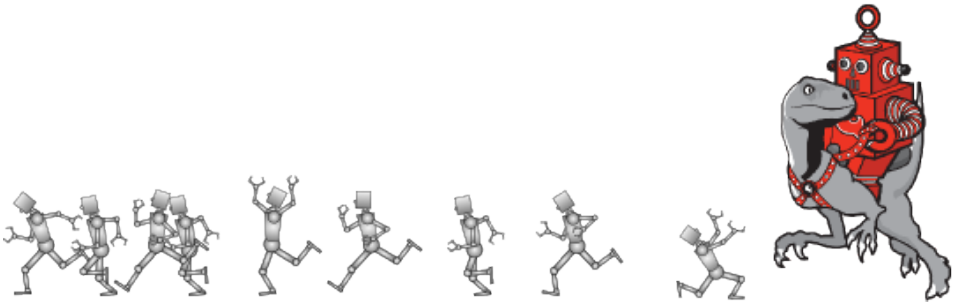
____

____

Now you're going to take that sentence and turn it into pseudo-code. Pay attention to where the parentheses and curly brackets are and how many there are. Start with writing the first question, put a curly bracket just after the question like this { and then put a curly bracket at the very end of the lines like this }. Now put what happens when the question is true and the second If statement inside of your first two curly brackets. If (you're confused) {look at example number two.}

____

____



SIK Worksheets v.1.0
Programming Concepts, If statements

Name:
Date:

Now that you understand the basics of If statements you're going to practice filling in various parts of some If statements. These If statements are not written in code, but you should be getting comfortable with what goes where as well as the parenthesis and curly brackets. Remember, you will only do what is in the curly brackets if the question is true. Fill in the blanks and if you feel like it make them funny.

If (____) { then you can fly. }

(your dog runs away) { then you need to go looking for your dog. }

If (you are hungry) { ____ }

If ____ { then you burp. }

If (you want to become an astronaut) { ____________________________ }

If (__________________________) { _________________________ }

(you build a robot) { _____________________________________ }

If (you build an electronic drum set) then you can practice quietly.

If (you are an elephant) { _____________________________________ }

(you make pancakes) _____________________________________

If (_________________________________) { you should hit the pinata. }

(you want pizza) ____

Programming Concepts, If statements

Purpose: Group activity teaching the concept of If statements and their syntax.
Materials: Cut up sheet of silly conditionals and actions.
Vocabulary to be explained prior to activity:

If statement: These simple statements exist in real life as well as in computer programming. They are simple statements that indicate if something is true or has occurred, then a resulting action takes place.
If statement pseudo-code: If (conditional) { action }

if: The word that always starts an If statement, convenient huh? Never capitalized.

Parenthesis ( ) : Indicates and bookends the conditional portion of an If statement.

Conditional: The question or condition that if true initiates the action of the If statement.

Curly brackets { } : Indicates and bookends the action portion of an If statement.

Action: Portion of code that occurs when the conditional is true. This can be anything including another If statement.

Activity:

Preparation: Cut up the conditional and action portions of the silly If statements included with this activity, or you can write your own and cut those up.

Activity: First mix and then distribute the slips of paper among your students. Explain the concept of an If statement to your students and then have them try to match up all the conditionals with the resulting actions. It is possible to mismatch the conditionals and actions, but this portion of the activity is mainly to have fun and establish the idea of a conditional and a resulting action, so don't worry if the kids mismatch some, just make sure you get some laughter out of this portion of the activity.

Second have seven students stand up to model portions of the If statement. The first student is the “If”, the second student is the first parenthesis, the third student is the conditional, the fourth student is the closing parenthesis, the fifth student is the first curly bracket, the sixth student is the resulting action and the final student is the closing curly bracket. Students then model one of the silly If statements they have matched up. Each student reads or says aloud the portion of the If statement they represent. Once the seven students have gone through the If statement, the last student sits down, all the standing students move one space over to the right and a new student stands up to join the group as the “If” portion. Students should cycle through this way until either everyone has had a turn to be each part of the If statement, or all the silly If statements have been used up. Encourage students who are representing the parenthesis and curly brackets to make parenthesis and curly brackets with their arms to demonstrate which are opening parenthesis and curly brackets and which are closing parenthesis and curly brackets.

Once the If statements and position of the parenthesis and brackets have been established in your classroom you can use the semantics where ever you see fit. For example, If (we line up quickly and quietly) { then we will have more recess time. }

SIK Worksheets v.1.0
Programming Concepts, Loops

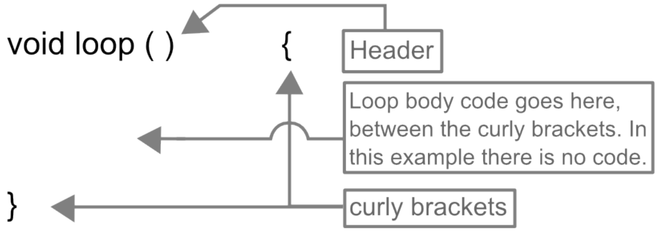
Name:
Date:

Vocabulary: repetition, header, loop body, curly brackets

In computer programming **repetition** means repeating a portion of code. This can happen in a bunch of different ways, but the most important thing is to first understand how it happens, not all the different ways it can happen. There are really only two portions to any **repetition**, the **header** and the **loop body**. The **header** usually looks about the same, but the **loop body** can contain any kind of code depending on what you are programming. The **loop body** can even contain another **repetition**!

Repetition with the **header, loop body,** semicolons and **curly brackets** labeled:

loop ( ):



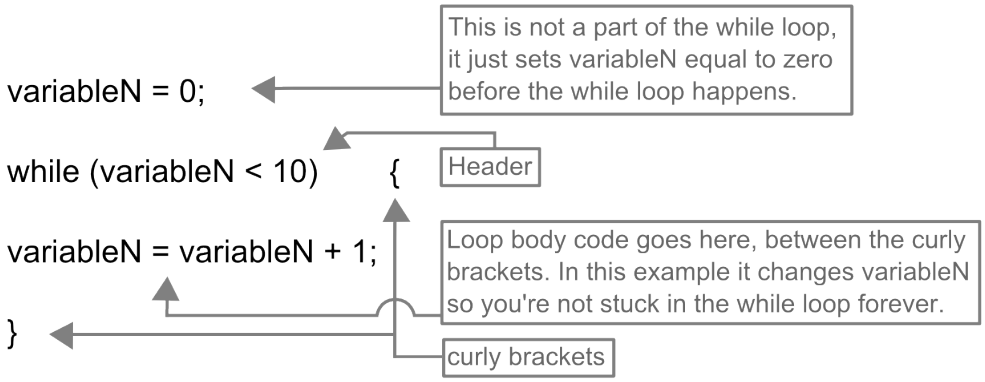
```
void loop ( )          {

}
```

Header

Loop body code goes here, between the curly brackets. In this example there is no code.

curly brackets

while ( ):



```
variableN = 0;

while (variableN < 10)       {

variableN = variableN + 1;

}
```

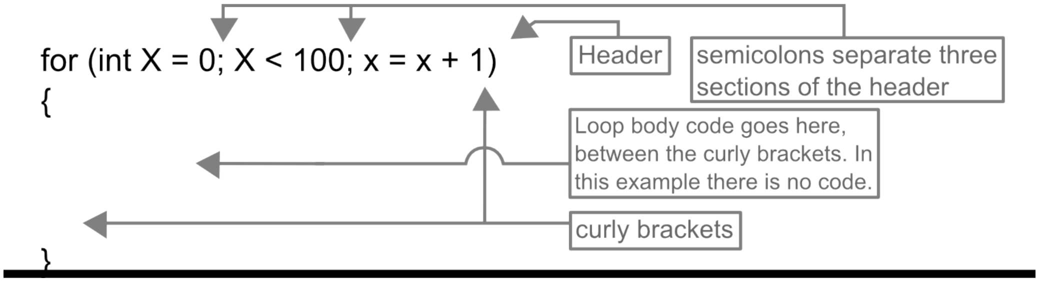
This is not a part of the while loop, it just sets variableN equal to zero before the while loop happens.

Header

Loop body code goes here, between the curly brackets. In this example it changes variableN so you're not stuck in the while loop forever.

curly brackets

for ( ):



```
for (int X = 0; X < 100; x = x + 1)
{

}
```

Header

semicolons separate three sections of the header

Loop body code goes here, between the curly brackets. In this example there is no code.

curly brackets

Just so we're clear on the important concepts that we will use when we talk about each different kind of **repetition,** please fill in definitions or explanations of the terms below.

Repetition: ____

Header: ____

Loop body: ____

Curly brackets: ____

(Use the back of the worksheet if necessary.)

SIK Worksheets v.1.0 **Name:**
Programming Concepts, Repetition **Date:**

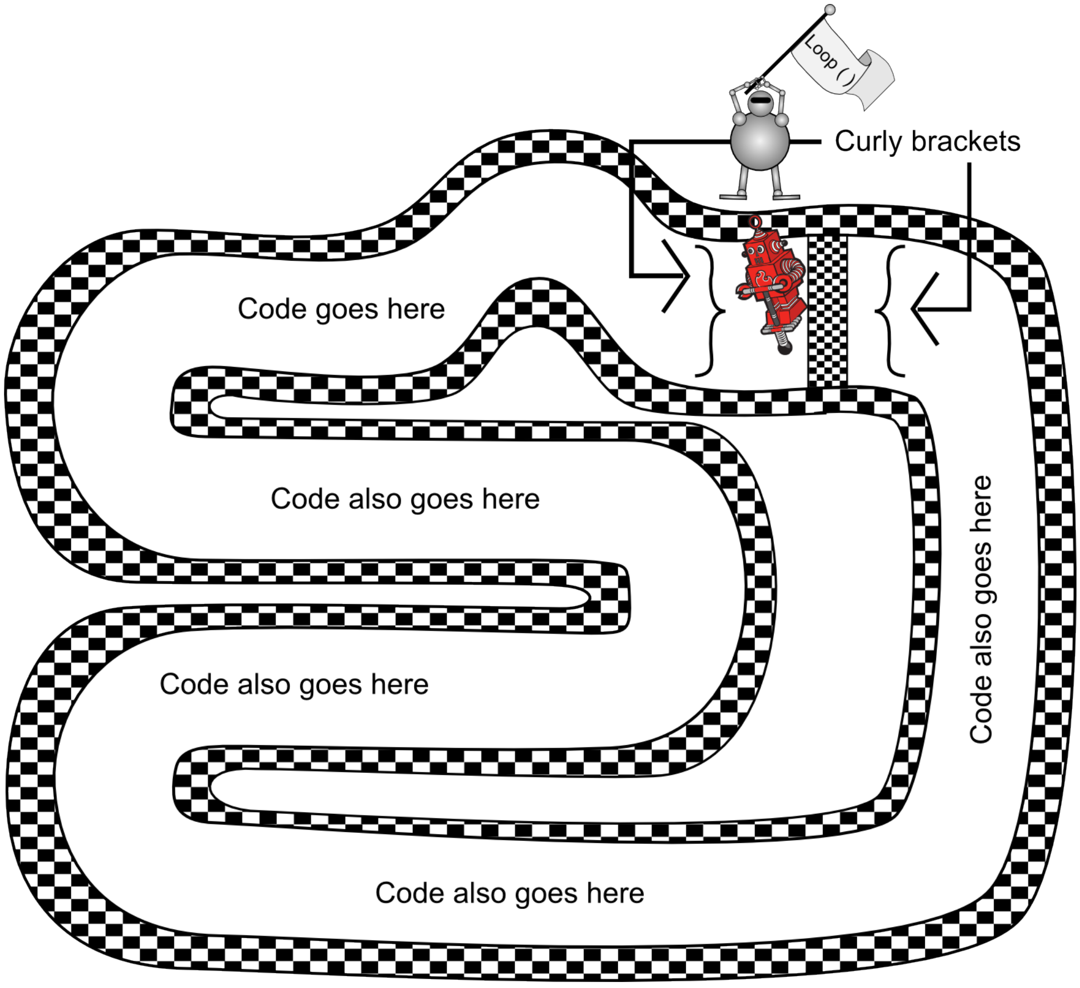
Vocabulary: loop ( )

The most common form of iteration in Arduino is called the `loop( )` function. It exists in all Arduino sketches and its whole purpose is to do all the code written inside of it once, then start over back at the beginning of the `loop( )` function and do it all again. Pretty simple, right? The most important things to remember about the `loop( )` function are that it is present in every single Arduino sketch, can only be used once per sketch, and it never ends. You will not find a single Arduino sketch that does not have a `loop( )`function in it and whenever anything happens in your sketch it is because of code inside the `loop( )` function.

The `loop( )` function looks like this:

```
void loop( ){
       // Lots (or just a little) of loop body code here between
            // curly brackets.
             }
```

Pay attention to the header and the curly brackets which are at the beginning and end of the loop body code. The header is just `void loop( )`. Think of the `loop( )` function as a racetrack. The `loop( )` header portion is the flag that starts the computer going around the racetrack and the curly brackets are the beginning and end of the racetrack. Now imagine your computer, Arduino, or robot running around and around the racetrack. It's up to you, the programmer, to put If statements, variables and other code along the way around the racetrack.



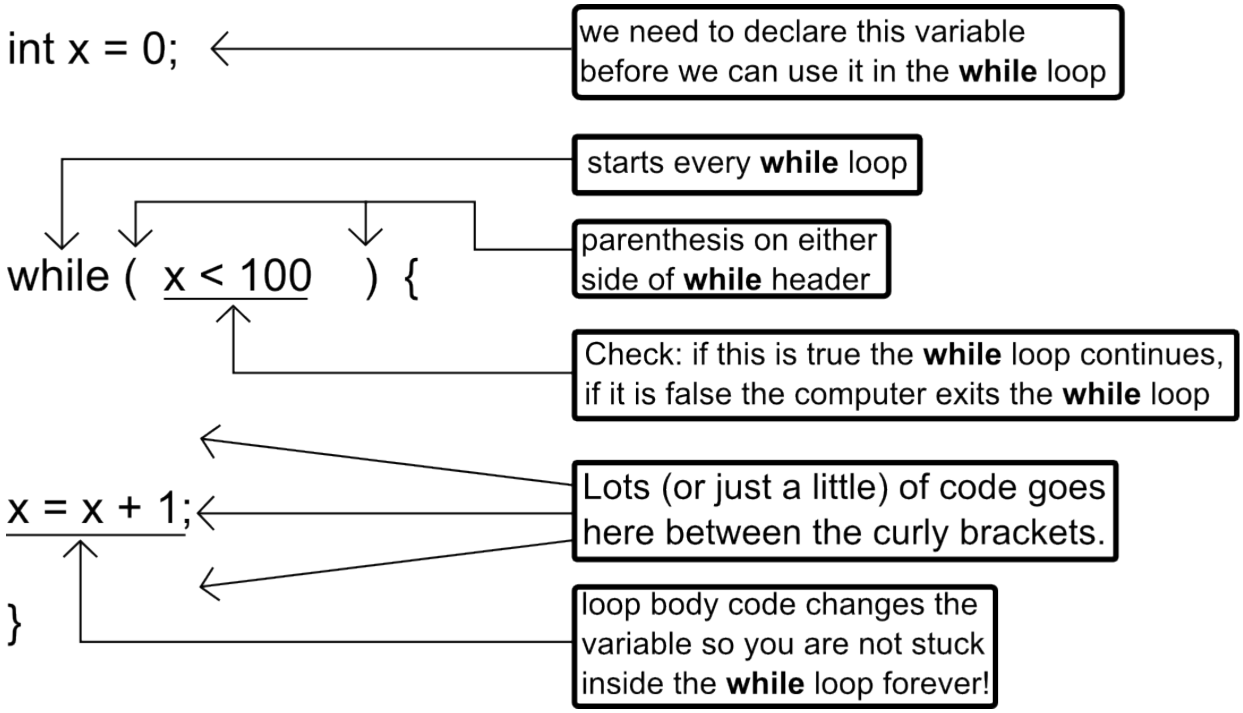
SIK Worksheets v.1.0 **Name:**
Programming Concepts, Repetition **Date:**

Vocabulary: while, loop ( )

So, you just learned about *`loop( )`*, which is the simplest form of repetition, but there are many other forms of repetition in Arduino. Another very common form of repetition is the `while` loop. A `while` loop is used when you want the computer or Arduino to do some code while a statement is true. The `while` loop is usually found inside of the *`loop( )`* function. The code of a `while` loop has two parts, the header and the loop body code. The header is the most important part to learn and always has the same structure. The code in the curly brackets below the header can be anything, it just depends on what you want to happen each time the computer goes around your `while` loop.

The header of a `while` loop has the word `while` and a statement inside of parenthesis. The `while` loop checks to see if the statement inside of the parenthesis is true and will repeat as long as that statement remains true. Pretty simple, right?

`While` loop example with variable declaration. Explanation of the `while` loop example.



```
int x = 0;

while (  x < 100    ) {

x = x + 1;

}
```

we need to declare this variable before we can use it in the **while** loop

starts every **while** loop

parenthesis on either side of **while** header

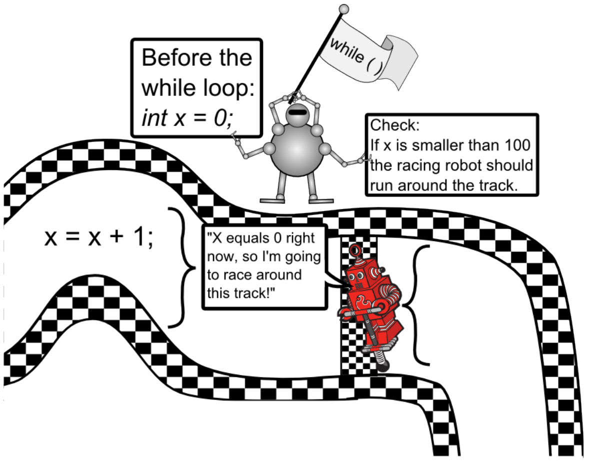
Check: if this is true the **while** loop continues, if it is false the computer exits the **while** loop

Lots (or just a little) of code goes here between the curly brackets.

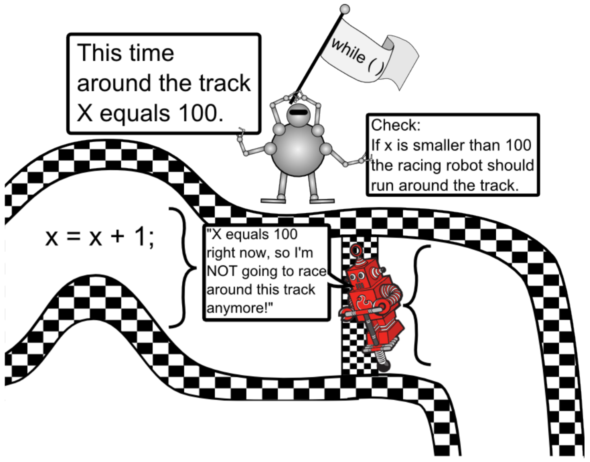
loop body code changes the variable so you are not stuck inside the **while** loop forever!

What happens during the `while` loop above using our robot racetrack as an example:

At the beginning:


Before the while loop: *int x = 0;*
while ( )
Check: If x is smaller than 100 the racing robot should run around the track.
x = x + 1;
"X equals 0 right now, so I'm going to race around this track!"

Later on, after 100 laps:

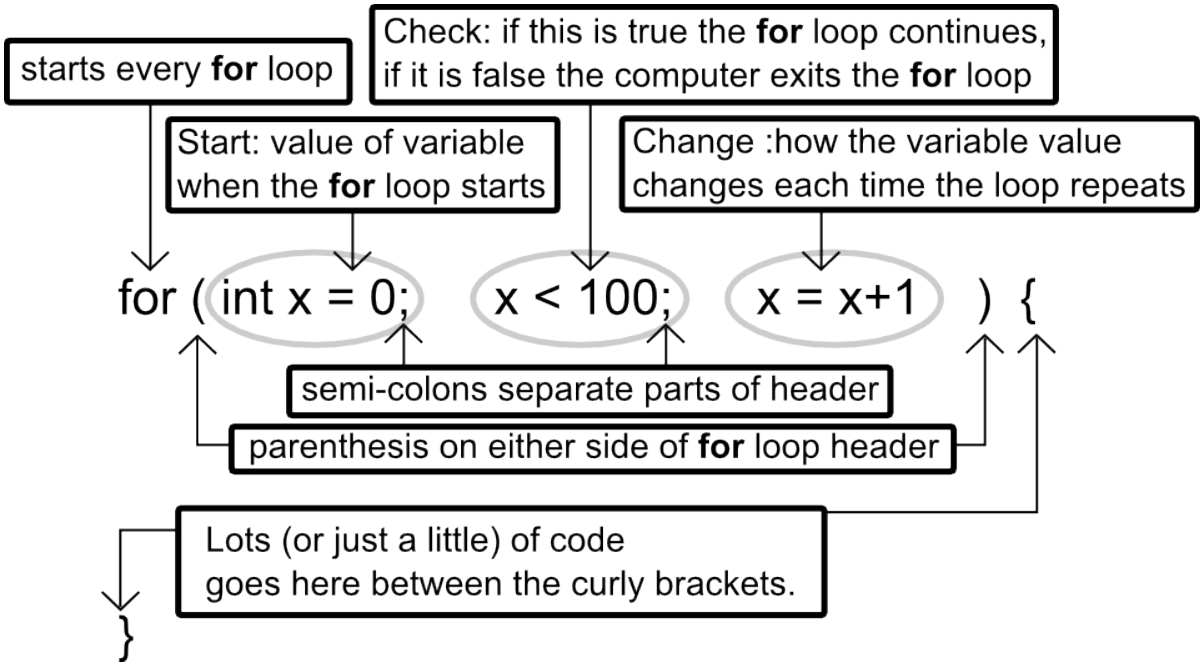

This time around the track X equals 100.
while ( )
Check: If x is smaller than 100 the racing robot should run around the track.
x = x + 1;
"X equals 100 right now, so I'm NOT going to race around this track anymore!"

SIK Worksheets v.1.0 **Name:**
Programming Concepts, Repetition **Date:**

Vocabulary: for, loop ( )

So, you just learned about `while`**,** which is a simple form of repetition, but there are many other loop functions. Another very common form of repetition is the `for` loop. A `for` loop is used when you want the computer or Arduino to change a variable each time through the loop and do code which often uses that variable. `For` loops are usually found inside of the *`loop( )`* function. The code of a `for` loop has two parts, the header and the code inside the loop. The header is the most important part to learn and always looks about the same. The loop body code in the curly brackets below the header can be anything, it just depends on what you want to happen each time the computer goes around your `for` loop.

The header of a `for` loop has the word `for` and in parenthesis three parts called **start**, **check** and **change**. Each of these parts have semicolons between them so you can tell them apart. These three parts (circled in gray below) are the most important parts to understand, they are the three simple parts you need to make a `for` loop work.


starts every **for** loop
Check: if this is true the **for** loop continues, if it is false the computer exits the **for** loop
Start: value of variable when the **for** loop starts
Change :how the variable value changes each time the loop repeats
for (int x = 0; x < 100; x = x+1) {
semi-colons separate parts of header
parenthesis on either side of **for** loop header
Lots (or just a little) of code goes here between the curly brackets.
}

Start:

The first circled part is **start**, this happens before anything else, it's sort of like putting on running shoes before starting to run around the track. It is a simple declaration and assignment of a variable, in this case the variable is an integer named x.

Check:

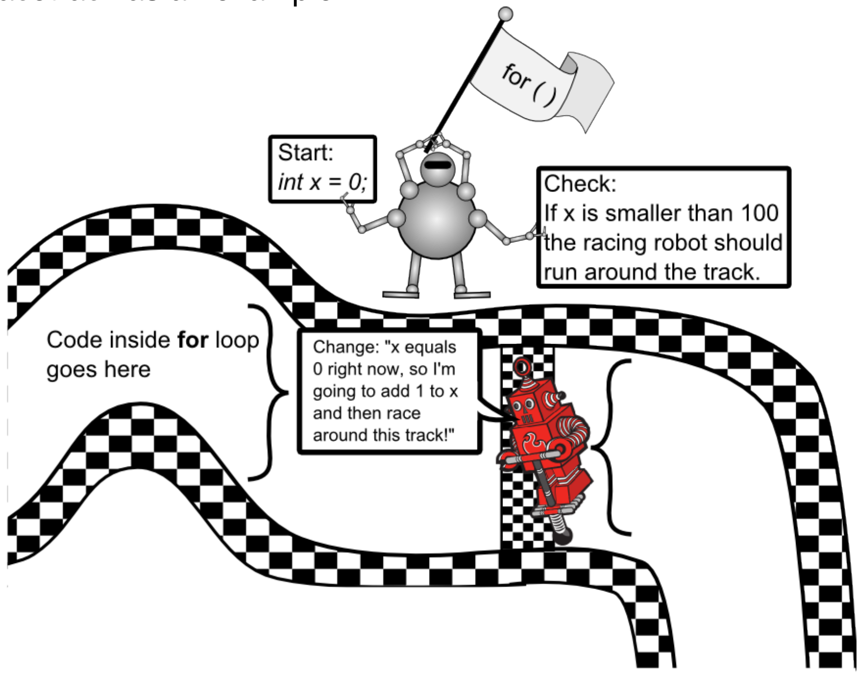
The second circled part is **check**. Every time the computer gets to the end of the `for` loop the computer will check to see if this part is true. The first time the `for` loop above checks, x is equal to zero, so the `for` loop continues, does **change** and then the code inside the curly brackets. It's kind of like checking how many laps a racer has completed to see if the racer has finished the race.

Change:

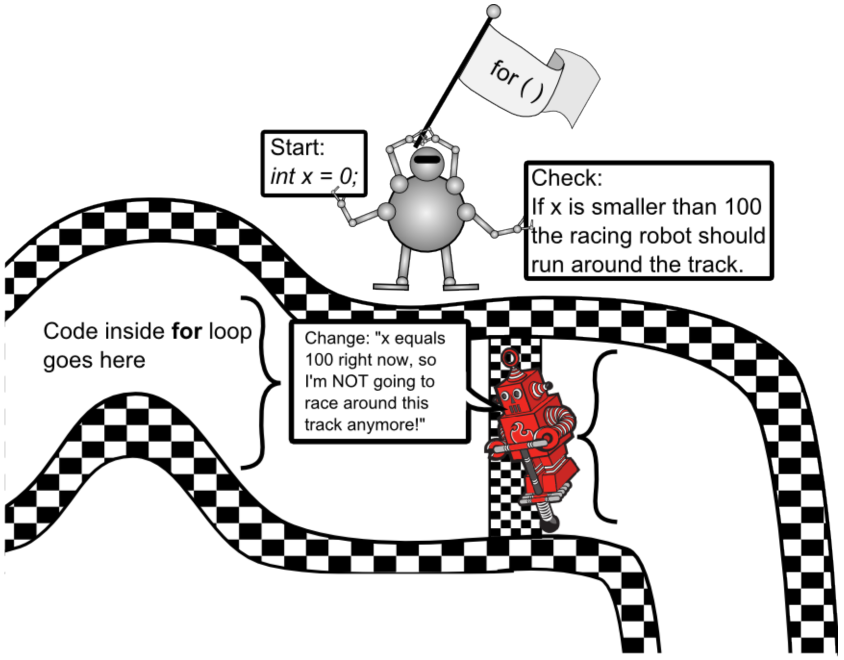
The third circled part is **change**, after the variable is checked it changes so that it is closer to making the **check** statement false so the `for` loop stops. For a racer this part of the `for` loop is like adding to (or updating) the number of laps or miles the racer has completed so far in the race.

Vocabulary: for, loop ( )

Here is an example of what happens when the `for` loop on the previous page begins using our robot racetrack as an example:


for ( )
Start:
int x = 0;
Check:
If x is smaller than 100 the racing robot should run around the track.
Code inside **for** loop goes here
Change: "x equals 0 right now, so I'm going to add 1 to x and then race around this track!"

The next time the racing robot makes its way around the track to the starting line it has to **check** again. It doesn't have to **start** again, but it does need to **check** to see if the race is over. The first time around the track, x will equal one and the **check** that x is smaller than 100 is still true. The robot **changes** the variable by adding one to x again (x now equals two) and then the robot runs around the track executing the loop body code between the curly brackets. The robot will continue to run around the racetrack until x equals 100 at which point the computer exits the `for` loop.


for ( )
Start:
int x = 0;
Check:
If x is smaller than 100 the racing robot should run around the track.
Code inside **for** loop goes here
Change: "x equals 100 right now, so I'm NOT going to race around this track anymore!"

SIK Worksheets v.1.0 **Name:**
Programming Concepts, Nested Repetition **Date:**

Vocabulary: Nested, Repetition

Now that you know about **repetition** we can talk about ways to put code inside of other code, which is called **nesting**, and in fact most loops are **nested** loops since they are inside of the original `loop( )` function. It's easy, all you do is put your loop inside the curly brackets of another loop. **Nested** if statements work exactly the same way as **nested** loops.

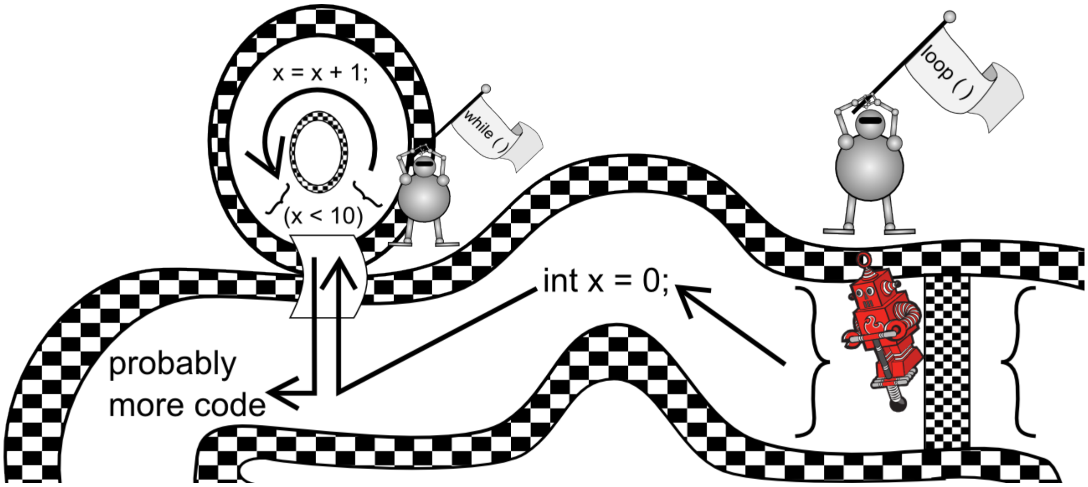
Example of **nested** loop:

```
void loop ( ) {
      int x = 0;
      while (x < 10) {
           x  =  x + 1;
      }
}
```

Example of **nested** if statement:

```
if (int x < 10 ) {
  if ( x == 5 ) {
  //code here happens if x < 10 & x = 5
  }
//code here happens if x < 10
}
```

Imagine your *`loop( )`* racetrack with another `for` loop racetrack attached to it. This way each time the robot runs (or drives or whatever) around the racetrack it must stop when it reaches a new while loop, run around that race track until that while loop is over and then it can continue running around the larger *`loop( )`* racetrack.



The robot has to run through the whole `while` loop before it can continue running around the larger loop racetrack. But let's break it down a little more; x starts as zero, if x is less than ten the robot continues running around the while loop until x is not less than ten. If the robot is adding one to x each time it checks the while loop then the robot must run around the while loop a total of ten times. The robot then exits the while loop and continues around the loop racetrack. Next time around the racetrack the variable x will be set to zero again just before the while loop. So, you don't have to worry about the while loop not working due to x being more than or equal to ten.

You can **nest** as many loops inside of other loops as you like, just make sure you don't get stuck inside of a loop. One way to do this is to misplace curly brackets, so make sure they're in the right spot. If this happens your computer or Arduino will just freeze and you won't really be able to tell why.

Nesting works for code other than just loops! You can **nest** if statements, loops and many other code structures. All you need to remember is that **nesting** is a complicated way to say “put code inside of other code” and that the computer eventually needs to get out of the **nested** statements and back to the *`loop( )`* function so everything can start over again.

Programming Concepts, Repetition

Purpose: Group activity teaching the concept of repetition as used in Arduino programming. Text `like this` denotes actual Arduino code.

Materials: Cones, large boards to display loop headers and pseudo-code, equipment for physical activities, and a field or gym.

Vocabulary to be explained prior to activity:

loop or repetition: A section of code that repeats.

repetition header: The line at the very beginning of a loop that tells the computer how the code inside the loop will repeat. This section is different for each different type of loop.

Conditional or question: This is the statement that is checked to see if the loop is completed. Conditionals are present in loop headers and often look like this: x < 10. This indicates that the loop will continue until x < 10 is false.

Increment: The section of code (may be in the header or may be in the loop body code) used to change the variable that is checked in the conditional. Using the example above, *x = x + 1*, one is added to x getting it a little closer to being larger than or equal to 10.

Nested repetition: A loop inside of a loop. This concept is key for any type of even slightly advanced programming.

Types of loops:

- **loop:** This loop is the most basic of all loops (that's why it's called loop) and is present in all Arduino sketches. `loop( )` repeats as long as there is power to the Arduino. Inside this form of repetition is where you will find all other forms of repetition.

Header: `loop( )`

Increment: N/A

Conditional: Power must be on.

- **while:** This loop repeats as long as the conditional listed inside the parenthesis is true. This loop's conditional is incremented in the body code or through an Arduino input.

Header: `while( )`

Increment: In body code or Arduino input

Conditional: Inside header parenthesis.

- **for:** This loop repeats as long as the conditional listed inside the parenthesis is true. The for loop header declares a variable, checks a conditional and increments the conditional variable all inside the parenthesis...

Header: `for ( int x = 0; x < 10; x = x + 1 )`

Increment: Inside header parenthesis, in this example x = x + 1.

Conditional: Inside header parenthesis, in this example x < 10.

Preparation: This activity is a physical activity and you will need to set up an obstacle loop or course that reflects the repetitions you have decided to include in this activity. You may wish to work with a gym teacher in order to set this activity up.

The examples in this activity require three different stations. These include a “loop” station at the beginning of the obstacle course with a teacher or student helper, a “while” station with jump ropes and an area for spinning in circles, and a “for” station with an area for doing jumping jacks and shooting basketballs.

Each station will need a poster displaying the pseudo-code that students need to follow in order to complete the obstacle loop. The poster materials are included with the rest of the activity materials in the folder programming in the file called LoopActivityMaterials.

You will also need a field for kids to run around or cones to set up an area for kids to run around inside a gym.

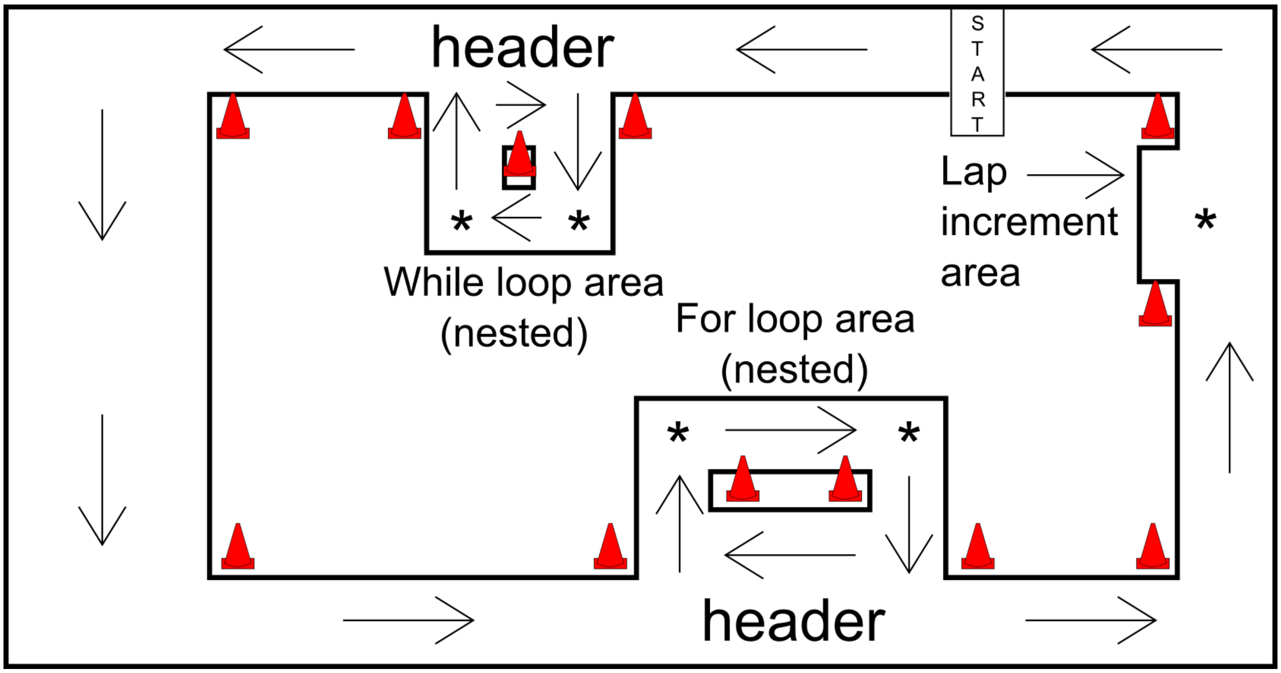
Also- this is a really big activity. It takes a lot of prep and will probably be chaos the first time you try it, but it is easily customizable to age or skill level and should be lots of fun if you stick with it.

Activity:

Students should have completed the introduction to repetition worksheets that come with this activity. Students should also be familiar with variables and if statements.

What your loop activity layout might look like:

▲	Cones kids have to run around
→	Direction kids should be running
*	Areas with physical activities that kids complete
header	Instructions in pseudo-code form for above activities



The idea is that the complete obstacle course from the start position back to the start position represents the `loop ( )` function. Inside this `loop ( )` function are two nested loops, a `while ( )` loop and a `for ( )` loop.

At the beginning of the obstacle course each student needs to declare an integer variable called lapNumber or something similar. This variable will be used in each of the loop activity areas and the lap increment area. The lapNumber variable can also be used to end the obstacle course if you do not wish to have students run the obstacle course until the end of the period. When students start the obstacle course lapNumber should equal zero since they have not run any laps yet.

Nested loop activity areas: These areas are nested loops where students will perform a certain number of tasks depending on what your loop headers say. You can have as many or as few activity areas as you like. You may also tailor the number of physical tasks inside these nested loops to make your obstacle course more fun for your students.

These activity areas should look like little loops that the students can run around completing tasks. The headers should follow the format of the loop type it represents. For examples see the end of the activity. Once inside the nested loop activity area students must complete the physical activities according to the pseudo-code posted inside the nested loop activity area. Once students are done with the first repetition of the physical activities inside the nested loop activity areas they should look at the header again and decide if they have completed the nested loop represented by the header. With younger students you may want to have someone helping them with this step. (This can be fun, the observer can yell out error in a friendly voice if students exit the loop too quickly) Once students have completed the nested loop activity area they continue around the obstacle course to the next activity.

Header examples: If a student's lapNumber is equal to three and the pseudo-code header reads:

```
while (lapNumber > basketsMade) {
do (lapNumber * 2) jump ropes at jump rope station
shoot lapNumber basketball baskets
}
```

This time around the obstacle course, the *s*tudent would run through the nested loop activity area once, jumping rope six times and shooting three baskets along the way.

If a student's lapNumber is equal to three and the pseudo-code header reads:

```
for (int x = 0; x < lapNumber * 2; x = x + 1) {
do (x * 2) jump ropes at jump rope station
shoot lapNumber basketball baskets
}
```

The student would run through this nested loop activity six times jumping rope a different amount and shooting three baskets each time for a total of thirty six jump ropes and eighteen baskets.

There is a lot of room for personalization in this activity; it's an opportunity to really solidify the loop concept as well as getting your kids some exercise.

Lap increment area: The lap increment area is where students will add one to their lapNumber variable to keep track of how many times they have run the obstacle course. You can also set up the headers and nested loop activities to use the lapNumber variable. The lap increment area is where you might insert an if statement to end the obstacle course after students complete a certain number of laps.

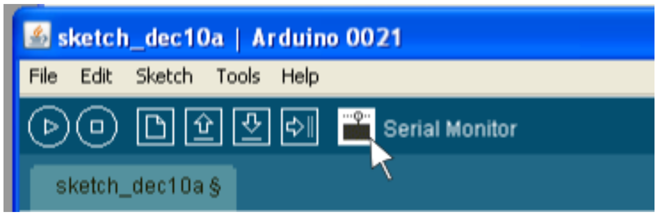
Additional thoughts: Definitely call the obstacle course a `loop( )` instead of an obstacle course in order to really get kids comfortable with the concepts. You may also wish to include your students in the planning of the obstacle course. Planning the obstacle course is another opportunity to talk about the loop concept and it gives them a stake in the learning exercise. Lastly, not that this needs pointing out, but this is a great activity just prior to computer lab time. Instead of having kids bouncing off the monitors they will be calmer and ready to sit still applying the concepts they just solidified through physical activity. This is great for kinesthetic learners in particular.

SIK Worksheets v.1.0 **Name:**
Serial Basics **Date:**

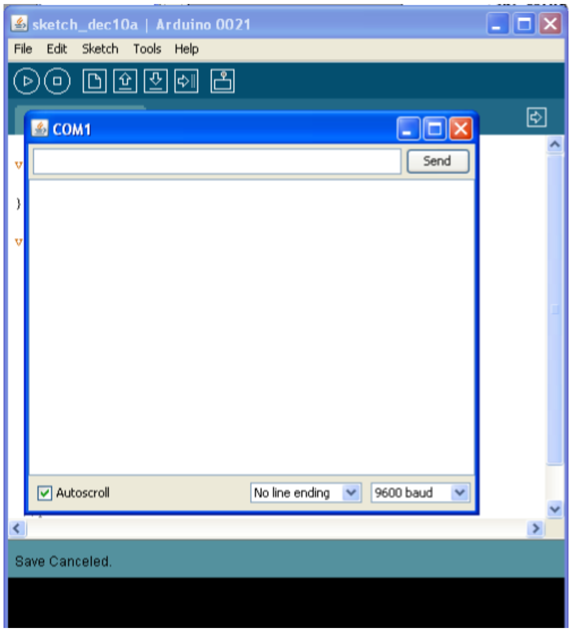
Serial is used to communicate between your computer and the Arduino as well as between Arduino boards and other devices. Serial uses a serial port (makes sense huh?) also known as UART, which stands for universal asynchronous receiver/transmitter to transmit and receive information. In this case the computer outputs Serial Communication via USB while the Arduino receives and transmits Serial using, you guessed it, the RX and TX pins. You use serial communication every time you upload code to your Arduino board. You will also use it to debug code and troubleshoot circuits. Basic serial communication is outlined in the following pages along with a simple activity to help you understand the concepts.

Serial Monitor: This is where you monitor your serial communication and set baud rate.

Activating the Serial Monitor:

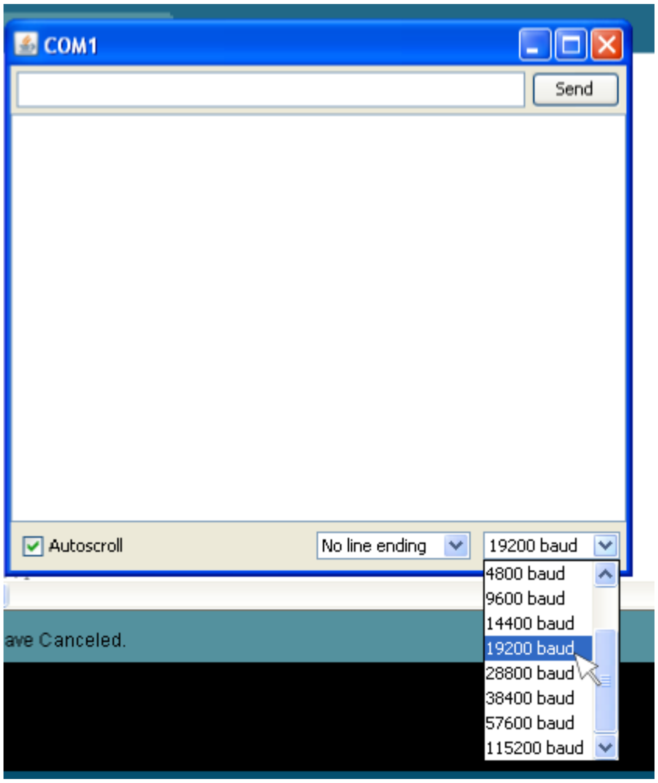


What the activated Serial Monitor looks like:



Save Canceled.

Setting the Serial Monitor baud rate:



ave Canceled.

There are many different baud rates, (9600 is the standard for Arduino) the higher the baud rate the faster the machines are communicating.

In the examples above there is no Serial communication taking place yet. When you are running code that uses Serial any messages or information you tell Serial to display will show up in the window that opens when you activate the monitor.

Things to remember about Serial from this page:

1. Serial is used to communicate, debug and troubleshoot.
2. Serial baud rate is the rate at which the machines communicate.

SIK Worksheets v.1.0 **Name:**
Serial Basics **Date:**

Serial setup:

The first thing you need to know to use Serial with your Arduino code is Serial setup. To setup Serial you simply type the following line inside your `setup( )` function:

`Serial.begin (9600);`

This line establishes that you are using the Digital Pins # 0 and # 1 on the Arduino for Serial communication. This means that you will not be able to use these pins as Input or Output because you are dedicating them to Serial communication. The number 9600 is the baud rate, this is the rate at which the computer and the Arduino communicate. You can change the baud rate depending on your needs but you need to make sure that the baud rate in your Serial setup and the baud rate on your Serial Monitor are the same. If your baud rates do not match up the Serial Monitor will display what appears to be gibberish, but is actually the correct communication incorrectly translated.

Using Serial for code debugging and circuit troubleshooting:

Once Serial is configured using the basic communication for debugging and troubleshooting is pretty easy. Anywhere in your sketch you wish the Arduino board to send a message type the line `Serial.println("communication here");`. This command will print whatever you type inside the quotation marks to the Serial Monitor followed by a return so that the next communication will print to the next line. If you wish to print something without the return use `Serial.print("communication here");`. To display the value of a variable using println simple remove the quotation marks and type the variable name inside the parenthesis. For example, type `Serial.println( i );` to display the value of the variable named `i`. This is useful in many different ways, if, for example, you wish to print some text followed by a variable or you want to display multiple variables before starting a new line in the Serial Monitor.

These lines are useful if you are trying to figure out what exactly your Arduino code is doing. Place a `println` command anywhere in the code, if the text in the `println` command shows up in your Serial Monitor you will know exactly when the Arduino reached that portion of code, if the text does not show up in the Serial Monitor you know that portion of code never executed and you need to rewrite.

To use Serial to troubleshoot a circuit use the `println` command just after reading an input or changing an output. This way you can print the value of a pin signal. For example, type `Serial.print("Analog pin 0 reads:");` and `Serial.println(analogRead(A0));` to display the signal on Analog Input Pin # 0. Replace the second portion with `Serial.println(digitalRead(10));` to display the signal on Digital Pin # 10.

Things to remember about Serial from this page:

1. If Serial is displaying gibberish check the baud rates.
2. Use `Serial.print("communication here");` to display text.
3. Use `Serial.println("communication here");` to display text and start a new line.
4. Use `Serial.print(variableName);` to display the value stored in variableName.
5. Use `Serial.print(digitalRead(10));` to display the state of Digital Pin # 10.

SIK Worksheets v.1.0 **Name:**
Serial Basics **Date:**

Using Serial for communication:

This is definitely beyond the scope of the S.I.K. but here are some basics for using Serial for device to device communication (other than your computer), not just debugging or troubleshooting. (The following paragraphs assume that you have Serial Communication hardware properly connected and powered on two different devices.)

First set up Serial as outlined on the previous page.

Use `Serial.println(“Outgoing communication here”);` to send information out on the transmit line.

When receiving communication the Serial commands get a little more complicated. First you need to tell the Arduino to listen for incoming communication. To do this you use the command `Serial.available();`, this command tells the computer how many bytes have been sent to the receive pin and are available for reading. The Serial receive buffer (computer speak for a temporary information storage space) can hold up to 128 bytes of information.

Once the Arduino knows that there is information available in the Serial receive buffer you can assign that information to a variable and then use the value of that variable to execute code. For example to assign the information in the Serial receive buffer to the variable `incomingByte` type the line; `incomingByte = Serial.read();` `Serial.read()` will only read the first available byte in the Serial receive buffer, so either use one byte communications or study up on parsing and string variable types. Below is an example of code that might be used to receive Serial communication at a baud rate of 9600.

```
//declare the variable incomingByte and assign it the value 0.
int incomingByte = 0;

void setup ( ) {
//establish serial communication at a baud rate of 9600
      Serial.begin(9600);
}
void loop ( ) {
//if there is information in the Serial receive buffer
      if (Serial.available() > 0){
//assign the first byte in buffer to incomingByte
            incomingByte = Serial.read();
      }
      if (incomingByte == 'A'){       //if incomingByte is A
            //execute code inside these brackets if incomingByte is A
      }
      if (incomingByte == 'B'){       //if incomingByte is B
            //execute code inside these brackets if incomingByte is B
      }
}
```

Additional things to note about Serial:

You cannot transmit and receive at the same time using Serial, you must do one or the other. You cannot hook more than two devices up to the same Serial line. In order to communicate between more than two devices you will need to use an Arduino library such as NewSoftSerial.

Things to remember about Serial from this page:

1. Serial communication requires knowing some code, but you can just look it up!
2. You cannot transmit and receive at the same time or hook up more than two devices.

SIK Worksheets v.1.0 **Name:**
Serial Debugging and Troubleshooting Activity **Date:**

Practicing simple Serial for debugging code:

If you would like to practice using Serial for code debugging open the code file Serial01, copy the text and paste it into an Arduino sketch. This sketch is mainly empty and waiting for you to add the Serial commands.

1. First type in the command line that begins Serial at a baud rate of your choosing below where the comment reads “`place serial setup here`”.
2. Next open the Serial Monitor and make sure it matches the baud rate you chose. This is an example of setting up Serial communication.
3. Next type a single Serial command that will display the text “Loop starts here” below the comment “`place serial statement 1 here`”. Make sure the command you use starts a new line after this text is displayed.
4. Then type a single Serial command that will display the text “Variable i is equal to “ below the comment “place serial statement 2 here”. Make sure you use the command that does not start a new line after this text is displayed.
5. Now add a command below this that will display the variable i. If you are having trouble with this portion don't forget that only text needs quotation marks around it for display in the Serial Monitor. This is an example of how you can use Serial communication to label your communication when you are trying to debug a troublesome variable.
6. Below the comment “`place serial statement 3 here`” add a command that will display the text “`this text displays when i is equal to 8`” and then start a new line. This is an example of how to display a variable value for debugging.
7. Below the comment “`place serial statement 4 here`” add a command that will display the text “`this text displays when i is equal to 9`” and then start a new line. This is an example of using Serial communication see if a portion of code ever actually executes.
8. Below the comment “`place serial statement 5 here`” add a command that will display the text “`Loop ends here`” and then start a new line.

SIK Worksheets v.1.0 **Name:**
Serial Debugging and Troubleshooting Activity **Date:**

Practicing simple Serial for troubleshooting circuits:

1. Open a sketch with the code for the S.I.K. circuit 07, build up this circuit on your breadboard.
2. Once the circuit is built add the command that establishes Serial communication in your `setup( )` function and make sure your baud rates match.
3. At the beginning of the `loop( )` function add the command line that will display the text "The pin " without starting a new line after.
4. Then add the command that will display the value of the variable "`buttonPin`" without starting a new line after.
5. Below this command add another command that will display the text " is " without starting a new line. That's right, just a space, the word "is" and another space, don't worry, we're almost done.
6. Now, below the comment that reads `//turn LED on`: add a command that displays *`digitalRead(buttonPin)`*, do the same below the comment that reads `//turn LED off`. For both of these commands make sure you use the form of Serial communication that starts a new line after displaying these values.

Ok, you're done and you should have a communication system that displays the electrical state of your button circuit. If there was a hardware issue somewhere in your circuit you could use this information to figure out what exactly is happening when you press the button.

To practice this same troubleshooting concept with analog inputs use the S.I.K. circuit 08. Follow the directions above but replace variable "`buttonPin`" with the variable "`sensorPin`" and replace the *`digitalRead(buttonPin)`* portions of your Serial communication commands with *`analogRead(sensorPin).`* These changes combined with the S.I.K. circuit 08 should make it clear how to troubleshoot a circuit with analog input.

To read output pins instead of input pins use the digitalRead command just like you did with the input pins.

Logic Flow Charts

Logic Flow Charts are a great way to sketch out how you want a circuit or chunk of code to act once it is completed. This way you can figure out how the whole project will act without getting distracted or confused by little details like electricity or programming. It's kind of like a game plan that a coach will put together before a game.

There are four major pieces that you will use over and over again when creating Logic Flow Charts. The four Logic Flow pieces are represented by a circle, a square, a diamond and lines connecting all the circles, squares and diamonds.

The **circle** is used to represent either a starting point, or a stopping point. This is easy to remember since you start every single Logic Flow Chart with a circle containing the word Start or Begin. Often you will end a Logic Flow Chart with an End or Finish circle, but sometimes there is no end to the chart and it simply begins again. This is the case with any circuits that never turn off, but are always on and collecting data.

The **square** is used to represent any action which has only one outcome. For example, when a video game console is turned on it always checks to see what video game is in it. It does this every time after it starts up and it never checks in a different way. This kind of action is represented by the square, it never changes and there is always only one outcome.

The **diamond** is used to represent a question or actions with more than one possible outcome. For example, once your video game has loaded there is often a menu with a bunch of options. This would be written in a Logic Flow Chart as a diamond with something like the words “Start Up Menu” written inside of it. Each action the user can take from this menu would be represented by lines coming off the diamond leading to another square, diamond, or circle. Maybe our example Logic Flow Chart would have three options leading away from the “Start Up Menu” diamond, one line to start a new game, one to continue a saved game and another for game settings. In the Logic Flow Chart each option is written beside the line leading away from the diamond. It is possible to have as many options as you like leading away from a diamond in a Logic Flow Chart.

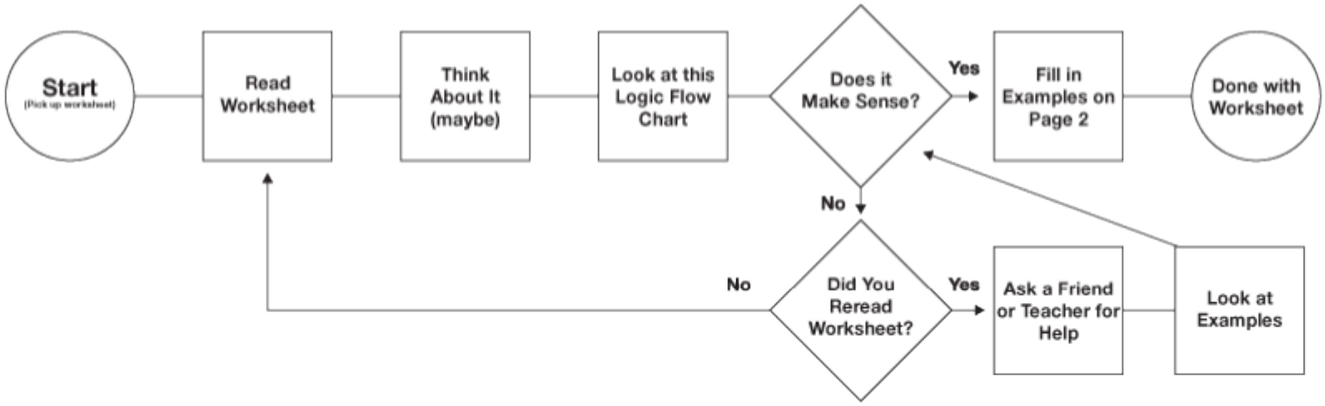
The **lines** in a Logic Flow Chart connect all the different pieces. These are there so the reader knows how to follow the Logic Flow Chart. The lines often have arrows on them and lead to whichever piece (circle, square, diamond) makes the most sense next. The lines usually have explanation of what has happened when they lead away from diamonds, so the reader knows which one to follow. Often some of these lines will run to a point closer to the beginning of the Logic Flow Chart. For example, the “Save Game” option might lead back to the “Start Up Menu” diamond, or it might lead straight to “Save and Quit”. It's up to you, you're the one making the Logic Flow Chart! All it has to do is make sense to you. Use the first Logic Flow Chart on the next page to help figure out how to use a Logic Flow Chart. Look at the second example, then complete the remaining Logic Flow Chart examples.

Name:
Date:

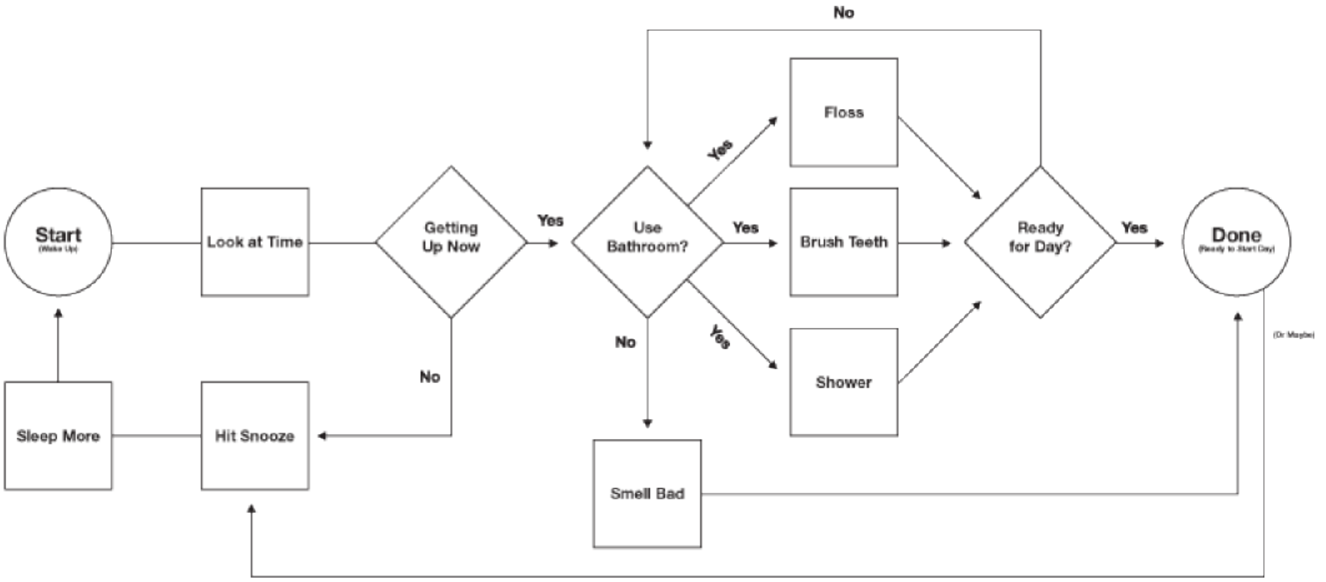
Logic Flow Charts II

Circles represent either start or end. **Squares** represent actions with one outcome.
Diamonds represent a question or action with multiple possible outcomes.
Lines and **arrows** represent logical paths between the **circles, squares** and **diamonds**.

Example 1:



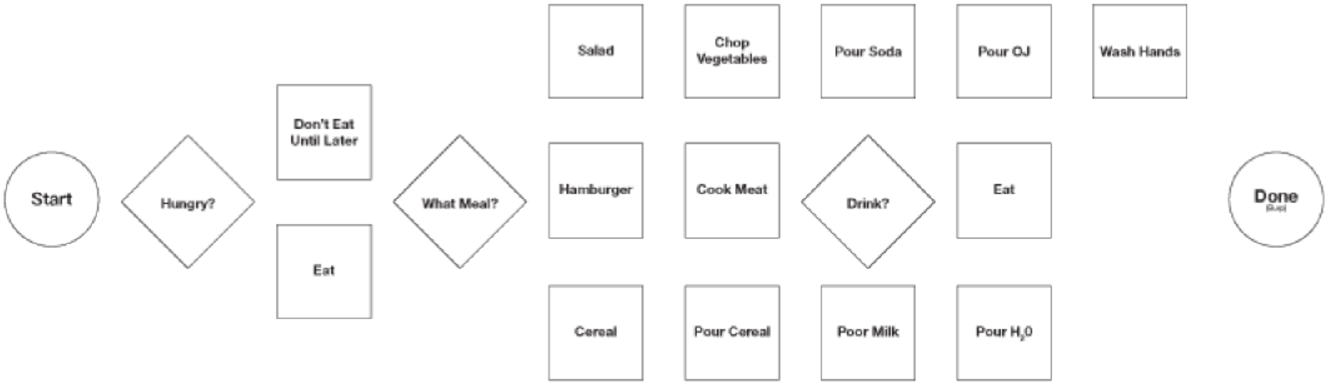
Example 2:



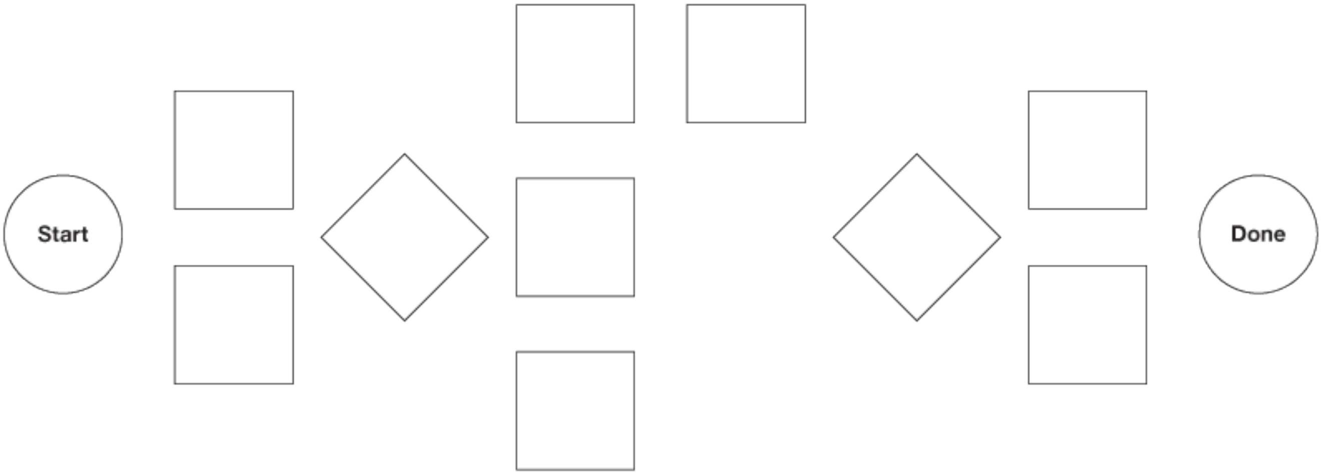
SIK Worksheets v.1.0 **Name:**
Logic Flow Charts **Date:**

Logic Flow Charts III

Example 3:
Fill in the lines and arrows.
There is no right answer,
but it must make sense.



Example 4:
Fill in the lines and text. Write outside of the boxes as necessary or use the back of the worksheet.

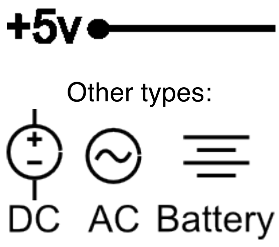
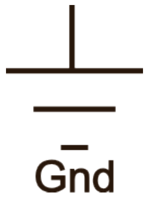
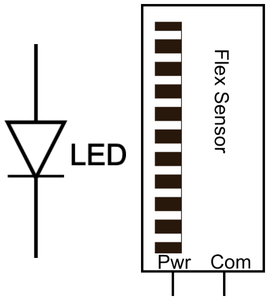
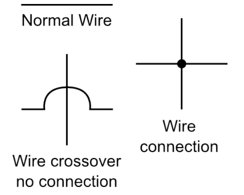


SIK Worksheets v.1.0 **Name:**
Schematics **Date:**

Schematics

An electrical schematic is a bunch of symbols representing one or more electrical circuits. Electrical schematics are a great way to sketch out the physical layout of a circuit. With a schematic it is possible to share circuit and prototypes ideas without giving away your electronics or creating an overlay. Being able to read schematics is definitely a useful skill anytime electronics are involved. Once you are able to read schematics, creating schematics just takes practice and a sharp eye when looking at wires and components.

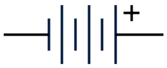
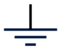
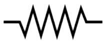
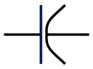
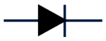
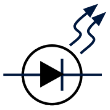
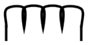
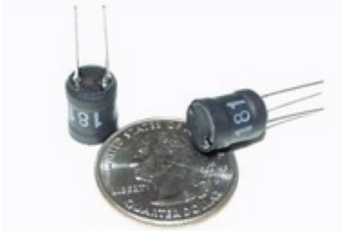
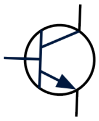
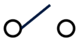
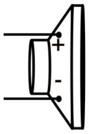
There are four major pieces that you will use over and over again when creating schematics. The four schematic pieces are **power source**, **ground**, **components** and the **wire** (or whatever your conductive connecting material is) connecting all the different parts. While schematics can be created by hand, these days electrical schematics are usually created using Electrical CAD software. Electrical CAD software is used to make sure that all schematics follow the same guidelines, making them easier to understand. While electrical schematics show how circuits are connected, they do not show what the completed circuit will look like. Schematics are guidelines, not physical representations of the circuits.

	The **power source** symbol is a small circle with the voltage written beside it. Power sources come in all sizes, but the S.I.K. mainly uses a 5V power source. Arduino output pins can also be used as power sources, so even though they are categorized as components (or at least a part of a component, the Arduino) they are often treated as power sources. Power sources are where the electricity necessary to make circuits work comes from.
	The **ground** symbol used in the S.I.K. is three horizontal lines, which decrease in width as they get closer to the bottom of the symbol with the letters Gnd beside or below them. In an electrical schematic ground can have a couple different types, this worksheet explains the most common, earth ground. An earth ground is a return path for electrical current as well as a reference point for measuring voltage. The term “earth ground” implies that the ground is a physical connection to the earth. This is sometimes true, but often ground is simply a connection to the lowest voltage value in a circuit or piece of equipment. The voltage value of ground will never change no matter how much electrical current it is absorbing.
	**Components** are represented by a bunch of different symbols. There are tons of different components with new types being invented every day! These are the parts of the circuit that use the electrical current to make stuff happen. This can be input or output, digital or analog, complicated (Arduino board) or very simple (resistor). These components can do many different things and it is important to understand the particular component in the schematic if you really want to know what the circuit is doing and how it uses electrical current.
 Normal Wire Wire crossover no connection Wire connection	**Wires** are represented by simple lines. The wires in a schematic connect all the different pieces. Pay attention to what wires look like in a schematic when they cross. If the wires are connected the lines will be straight with a circle where they cross, if the wires are not connected one of the wires will form a semi-circle where the lines cross.

SIK Worksheets v.1.0
Schematics

Name:
Date:

Common Schematic Symbols

Power / Battery / Cell			
Ground			
Resistor (Colored bands indicate resistive strength - see cheatsheet for details)			
Capacitor (Electrolytic on left; Ceramic on right)			
Diode			
Light Emitting Diode (LED)			
Inductor			
Transistor (arrow pointing out is NPN, pointing in is PNP)			
Switch			
Speaker			

SIK Worksheets v.1.0
Circuit 1, Middle School

Name:
Date:

Ohm's Law: V = I * R I = V / R R = V / I

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

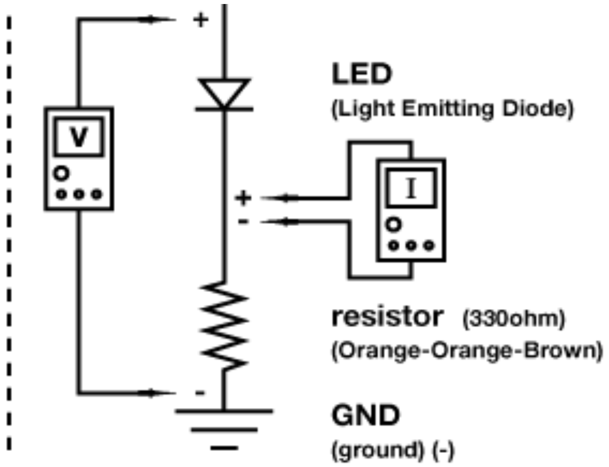
Did you get your LED turned on? Make sure it's red. Switch the Arduino pin # 9 with 5v on the Arduino.

Give values for Voltage, Current and Resistance for each LED circuit setup. Find Resistance with Ohm's Law. Hint: Break the circuit between the Arduino pin and LED to measure the current.

330Ω (circuit as is):
V = ______v I = ______mA R = ______Ω
With two LEDs and 330Ω :
V = ______v I = ______mA R = ______Ω
One LED and 10KΩ resistor:
V = ______v I = ______mA R = ______Ω
Two LEDs and 10KΩ resistor:
V = ______v I = ______mA R = ______Ω

Circle the Ground.
Draw arrows to indicate direction of current on dotted line.
Add an on/off switch to this schematic.

For questions that use multiple LEDs make sure you place additional LEDs between the multimeter leads.



Draw a logic flow chart of the circuit here:
Use the back of this worksheet if necessary.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

What circuits or projects would you like to add LEDs to? List at least three reasons you might add LEDs to an existing circuit or product that you might use. For example: to indicate when a squirt gun is running low on water or to add a flashlight to your hat.

SIK Worksheets v.1.0
Circuit 2, Middle School

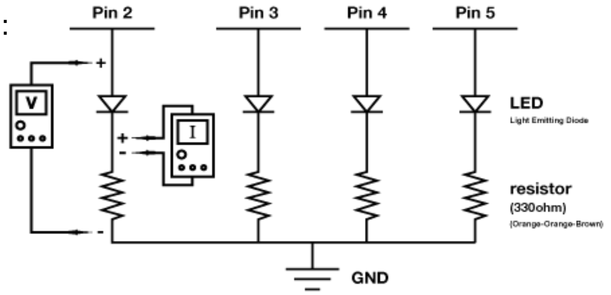
Name:
Date:

Ohm's Law: V = I * R I = V / R R = V / I

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Got your LEDs turned on? Great. Load the Circ02Expansion Code. In this code the last LED pin is an analog output using PWM (Pulse Width Modulation). Make sure to use a red LED.

Give values for Voltage, Current and Resistance for each LED circuit setup. Find Resistance with Ohm's Law. Hint: Break the circuit between Arduino pin and LED to measure the current.



PWM @ 63.75 (or 25%)

V = ______v I = ______mA R = ______Ω

PWM @ 127.5 (or 50%)

V = ______v I = ______mA R = ______Ω

PWM @ 191.25 (or 75%)

V = ______v I = ______mA R = ______Ω

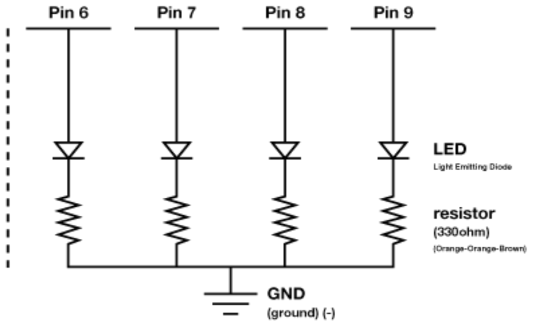
PWM @ 255 (or 100%)

V = ______v I = ______mA R = ______Ω

Circle all the Power Sources (This one is a little trickier)

Draw arrows to indicate direction of current on dotted line.

Add an on/off switch for one LED to this schematic.



Draw a logic flow chart of the circuit here:
Use the back of this worksheet if necessary.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

What circuits or projects would you like to add LEDs to? Can you think of at least three reasons you might add multiple LEDs to an existing circuit or product that you would use? For example: clock that shuts off an LED every time you are done with a class, turning off all the LEDs by the end of the day so you know you are free.

SIK Worksheets v.1.0
Circuit 3, Middle School

Name:
Date:

Ohm's Law: V = I * R I = V / R R = V / I

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

____

____

____

____

Got your motor running? Great. Load the Circ03Expansion Code. Fill in the answers below.

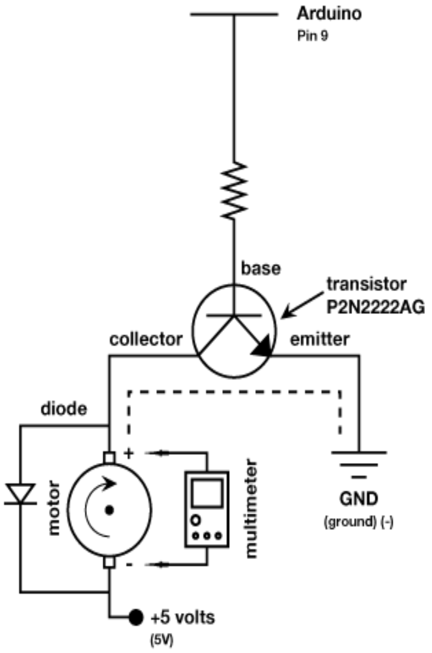
Give values for Voltage, Current and Resistance for each motor value. Find Current by breaking the circuit and using your multimeter. Calculate the Resistance using Ohm's Law. Record all values to the hundredths place.

Motor 50%
V = ______v I = ______mA R = ______Ω
Motor 60%
V = ______v I = ______mA R = ______Ω
Motor 75%
V = ______v I = ______mA R = ______Ω
Motor 100%
V = ______v I = ______mA R = ______Ω

Circle the diode.

Draw arrows to indicate direction of current on dotted line.

Add an on/off switch to this schematic.



Draw a logic flow chart of the circuit here:
Use the back of this worksheet if necessary.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

How would you use this circuit if you were an engineer? Would you make a break-dancing robot penguin? To move a trapdoor? To make a yo-yo that plays itself? Get creative.

____

____

____

____

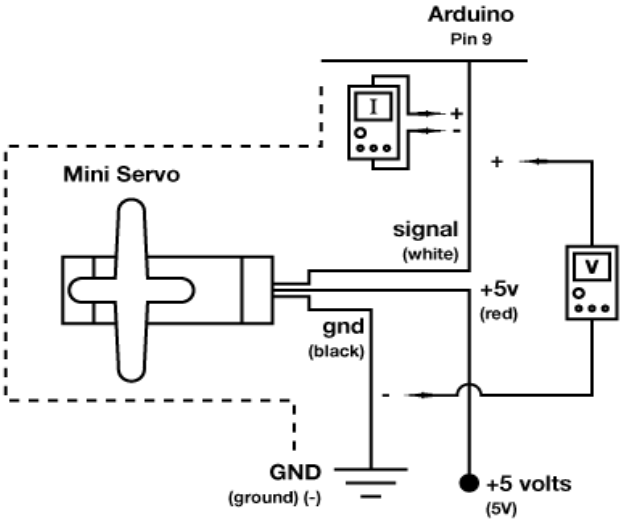
SIK Worksheets v.1.0
Circuit 4, Middle School

Name:
Date:

Ohm's Law: V = I * R I = V / R R = V / I

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Got your servo running? Great.



Give values for Voltage, Current and Resistance for the pin # 9 value while the servo is in motion. Find the current by breaking the circuit and measuring at multimeter with I. Find resistance using Ohm's Law.

Highest reading while Servo is in motion:

V = ______v I = ______mA R = ______Ω

What does the Arduino pin # 9 do in this circuit?

Draw arrows to indicate direction of current on the dotted line.

Add an on/off switch to this schematic.

Draw a logic flow chart of the circuit here:
Use the back of this worksheet if necessary.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

A Servo can't rotate continuously more than 360 degrees, as opposed to a motor which can turn all the way past 360 degrees as many times as you like. However, a servo remembers what its position is while a motor only knows if it is running forward or backwards. Can you think of any situations in which you would need a Servo instead of a motor? How about the other way around? Write three examples, at least one of each, below.

SIK Worksheets v.1.0
Circuit 5, Middle School

Name:
Date:

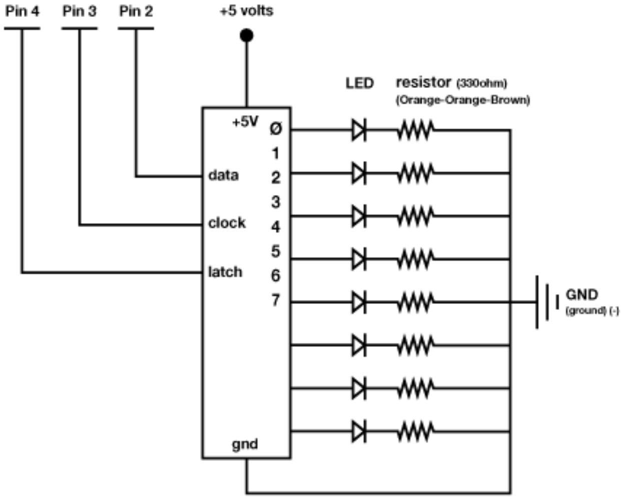
Shift Registers are used to control multiple pins using only three input pins to set the output pins. This can be useful if you want to control more than three objects using only three pins (as long as they always operate in the same order). What objects would you control using a shift register? List at least four and make sure the objects make sense together. Ex: A waffle iron, an eggbeater, a servo to pour the batter, and a timer.

What does the Arduino pin # 2 do in this circuit?

What does the Arduino pin # 3 do in this circuit?

What does the Arduino pin # 4 do in this circuit?

If all the LEDs are turned on, what would have to happen in order for LED # 5 to turn off?



Draw a logic flow chart of the circuit here: You can limit the diagram to one of the LEDs. Use the back of this worksheet if necessary.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

Using the circuit exactly as it is, with eight LEDs, what applications can you think of for the shift register? List at least three and explain what each LED would indicate.

SIK Worksheets v.1.0
Circuit 6, Middle School

Name:
Date:

Ohm's Law: **V = I * R** **I = V / R** **R = V / I**

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Got the annoying song blaring out of your speaker? Upload Circ06Expansion Code to your Arduino.

Give values for Voltage, Current and Resistance for each note value. Find Current by breaking the circuit and using your multimeter. Record voltage to the thousandths place. Calculate Resistance using Ohm's Law.

Note A: V =______v I =______mA R =______Ω

Note C: V =______v I =______mA R =______Ω

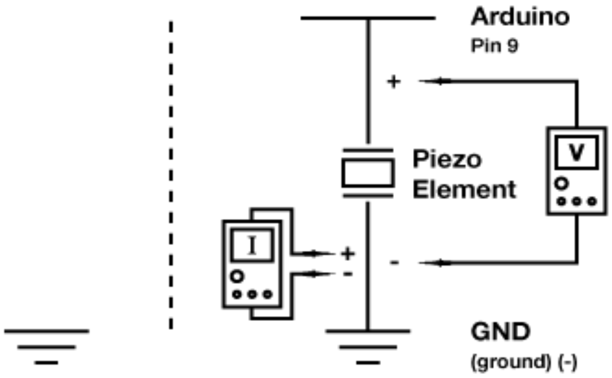
Note E: V =______v I =______mA R =______Ω

Note G: V =______v I =______mA R =______Ω

What does the Arduino pin # 9 do in this circuit?

____

Draw a logic flow chart of the circuit here:
Use the back of this worksheet if necessary.



Draw arrows indicating current direction on dotted line.
Add another Piezo Element to the schematic so you can write harmonies. Be sure to show which Arduino pin you will attach it to.
Add an on/off switch to this schematic.

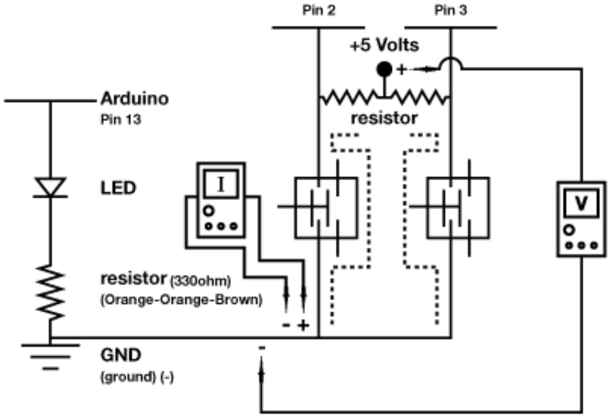
Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

Other than annoying your friends, how could you use the Piezo Element in a project? Example: create a timer that plays an annoying song faster and faster as time runs out. Write at least two examples.

SIK Worksheets v.1.0 **Name:**
Circuit 7, Middle School **Date:**

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Can you turn your LED on and off using both buttons? Great. Pay attention to this circuit, buttons are one of the most basic forms of user interface.



Give values for Voltage, Current and Resistance for each question. Find Current either by breaking the circuit and/or using your multimeter.

Button on pin 2 pushed:

V = ______v I = _______mA R = ______Ω
Button on pin 3 pushed:

V = ______v I = _______mA R = ______Ω

Think about the question above. What makes the LED turn on and off?

If the resistor's value is 10000Ω, what is the resistance of the button? _________________Ω

Replace the LED component (in the space below the schematic to the right) with an element or component from one of the previous circuits. Extra credit if you decide to replace it with a motor.

Draw a logic flow chart of the circuit here:
Use the back of this worksheet if necessary.

Draw arrows to indicate direction of current on the dotted line.
Circle the resistor with the largest value.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

Buttons are everywhere. List at least two different kinds of buttons that you might not think of as being buttons. Examples: piano keys and go Google “reed switches”. Now list at least two items that are not technically buttons, but could be used as buttons. Example: snaps on a shirt.

SIK Worksheets v.1.0 **Name:**
Circuit 8, Middle School **Date:**

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

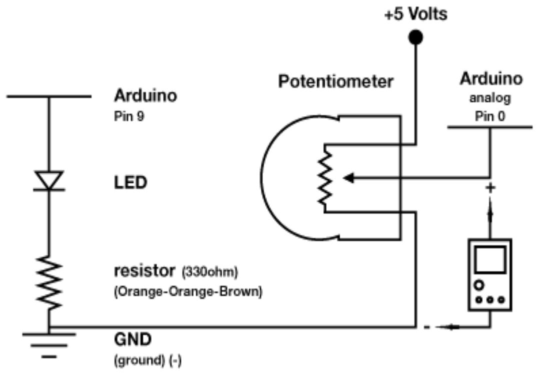
Can you turn your LED up and down using the potentiometer? Great. Pay attention to this circuit, potentiometers (also called trimpots) are great for creating analog user interfaces. With a potentiometer there are up to 1024 settings on a single dial!

Add the following to the circuit code and upload:
In Setup: `Serial.begin(9600);`
In Loop after all other code:
`Serial.println(sensorValue);`
Now open the Serial Communication window.

Replace the LED component (in the schematic) with an element or component from one of the previous circuits, extra credit if you decide to replace it with a motor. Explain which aspect of the element or component is controlled by the potentiometer. Example: replace LED with Piezo speaker and control pitch with potentiometer.

Aspect:

Draw a logic flow chart of the circuit (with Serial): Use the back of this worksheet if necessary.



Use your multimeter as indicated and measure the voltage of the potentiometer circuit while you turn the dial up and down. Explain below what happens.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

Potentiometers are everywhere. List at least three appliances that use potentiometers as an input. Also list what the potentiometer input controls (also known as an output).

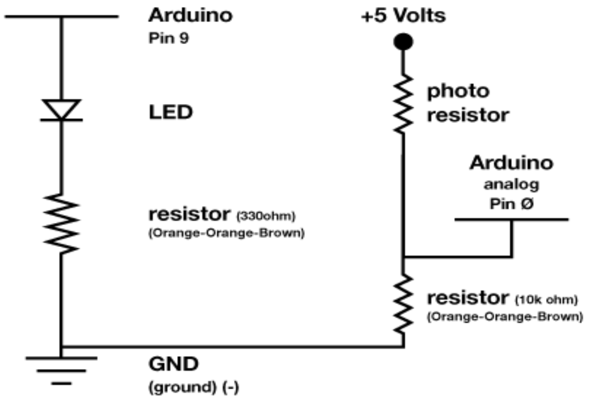
SIK Worksheets v.1.0 **Name:**
Circuit 9, Middle School **Date:**

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Can you turn your LED up and down using the photoresistor? Great.

Add the following to the circuit code and upload:
In Setup: `Serial.begin(9600);`
In Loop after all other code:
`Serial.println(lightValue);`
Now open the Serial Communication window.
Replace the LED component (in the space below the schematic on the right) with an element or component from one of the previous circuits, extra credit if you decide to replace it with a motor and do so correctly.
Make sure an aspect of the element or component is controlled by the photoresistor. Example: replace LED with Piezo and control pitch with potentiometer. Aspect controlled:

Why does this circuit use an analog pin as an input?



Circle all the resistors.

Draw a logic flow chart of the circuit (with Serial):
Use the back of this worksheet if necessary.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

The output variable lightValue can go all the way up to 900 but your LED input can only go up to 255. What word in the code fixes this and how would you describe this action in a mathematical sense?

SIK Worksheets v.1.0 **Name:**
Circuit 10, Middle School **Date:**

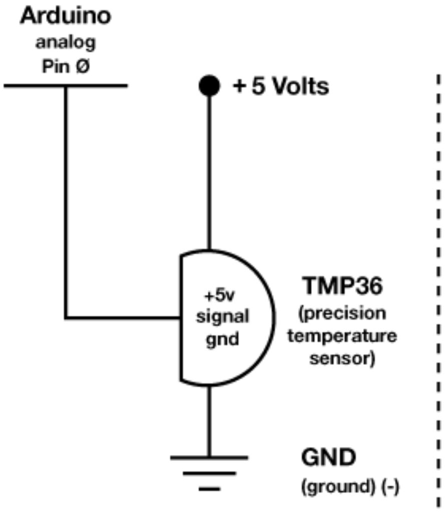
How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Does your temperature sensor work? Great.

What line in the code displays the temperature?

What other line in the code is necessary to establish communication with your computer so it can display the temperature?

Upload the Circ10Expansion Code to your Arduino, then add an LED and a resistor to the circuit. Control the LED's brightness with the temperature sensor. By now you should be able to do this with no help, but here's a hint anyways: PWM pins = 3, 5, 6, 9, 10, 11



Draw arrows on the dotted line to indicate direction of current flow.

Draw a logic flow chart of the expanded circuit:
Use the back of this worksheet if necessary.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

What ways, other than controlling an air conditioner, could a temperature sensor be useful? List at least three and explain what is controlled by the temperature sensor in each.

SIK Worksheets v.1.0
Circuit 11, Middle School

Name:
Date:

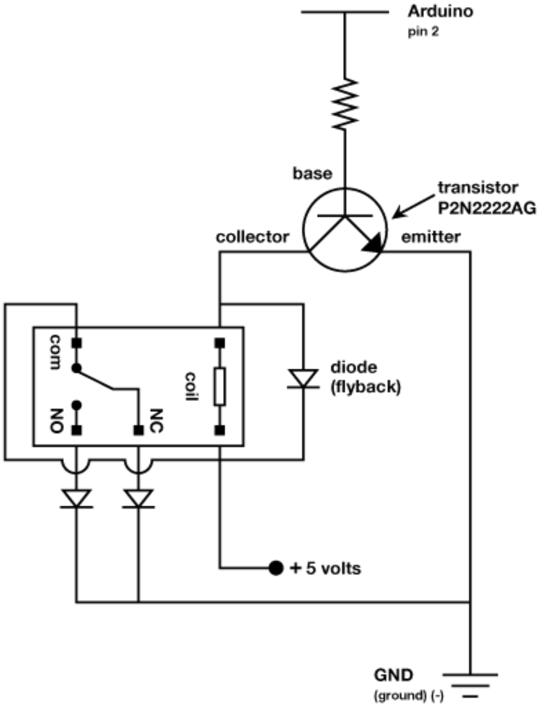
How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Does your relay work? Great. Upload Circ11Expansion Code to your Arduino, then add a button to your circuit and to the right of the schematic, make sure you include a pull up resistor (10KΩ resistor).

Name one thing you would have to change to use a potentiometer to control the relay instead of a button, extra credit for writing what you would need to add in the code as well. Use English, not code.

If the NO in the relay stands for Normally Open, what do you think NC stands for?

Circle the transistor.
Circle all diodes.



Draw a logic flow chart of the expanded circuit :
Use the back of this worksheet if necessary.

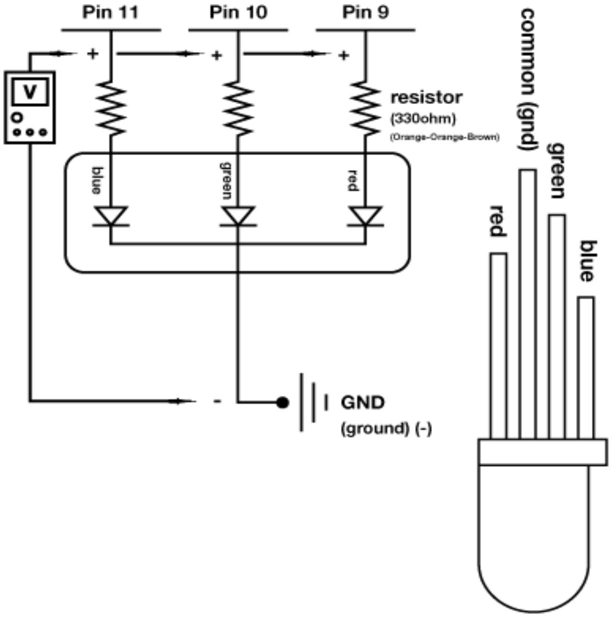
Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

Imagine your perfect relay machine. What two things would your relay switch back and forth between and why? Get creative. Example: a soft serve ice cream machine and an electric caramel pump so that making sundaes would be a little less work and give you more time to eat them.

SIK Worksheets v.1.0 **Name:**
Circuit 12, Middle School **Date:**

This LED has three inputs and three outputs. What other circuits have three inputs? Three outputs? Provide an example of each.

Does the RGB LED work? Great. Upload the Circ12Expansion Code to your Arduino, add a trimpot, a temperature sensor and a photoresistor to the circuit. Connect the trimpot to analog pin 1, the temperature sensor to analog pin 2, and the photoresistor to analog pin 0. Use some of the previous circuit schematics if you get stumped. You may also switch out the temperature sensor for the flex sensor or soft potentiometer so you have more control of the RGB LED if you like, but you will also need to change the code a little.



Connect a multimeter to each line that is connected to an Arduino pin. Notice how the voltage changes while you use the sensor or interface coupled with each pin.
What should the voltage values for each pin be to make the RGB LED as red as it will get?

Pin 11 V =____v Pin 10 V =____v Pin 9 V =____v

What does “RGB LED” stand for?

What do you think “RGBY LED” stands for?

Add an On/Off switch to your schematic.

Draw a logic flow chart of the expanded circuit: Use the back of this worksheet if necessary.

Draw one example of how this circuit could be used in everyday life. Label all components and give it a title.

Other than making projects pretty, what are some possible uses for a RGB LED? List at least three.

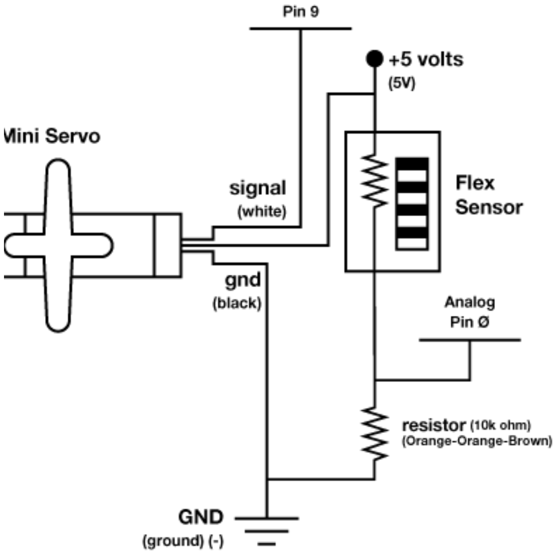
SIK Worksheets v.1.0 **Name:**
Circuit 13, Middle School **Date:**

Now we're starting to work with some more complicated sensors. The flex sensor has tons of real world applications. List three and explain why you can't use a regular potentiometer instead of a flex sensor. Example: use the sensor to measure the flex on a fishing pole and cut the line if the pole ever comes close to breaking. You could not use a potentiometer because it would be difficult to attach it.

Got your flex sensor and servo working? Great, but what if you want to measure flex in both directions? Add the necessary components to the schematic below (add components on right) and describe (in plain English) what you would need to add to the code to keep the single servo as your output with your new schematic.

Unplug the flex sensor completely and look at your Serial Communication window. You should still be getting some values even though there is no sensor plugged in. This is due to something called “float” which occurs when an Arduino pin is expecting input but there is no sensor attached to it. What is the highest value you receive and why is it important to know about float?

What if you wanted your flex sensor to measure a smaller range of flexing (because what you are measuring is less flexible), but you want the same range of motion for your servo, how could you make it do that?



Add an On/Off switch to your schematic.

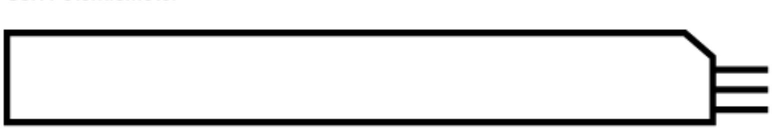
Circle any Arduino pins that take input on your modified schematic.

Imagine your flex sensor is thirty feet long. List at least three things you could do with it.

SIK Worksheets v.1.0 **Name:**
Circuit 14, Middle School **Date:**

The soft potentiometer is touchy, sometimes you will notice incorrect readings due to how you touch the sensor. The RGB will even change a little just before you touch the sensor! Explain how this alters the ways in which you can use this sensor. Explain at least one possible fix or work around.

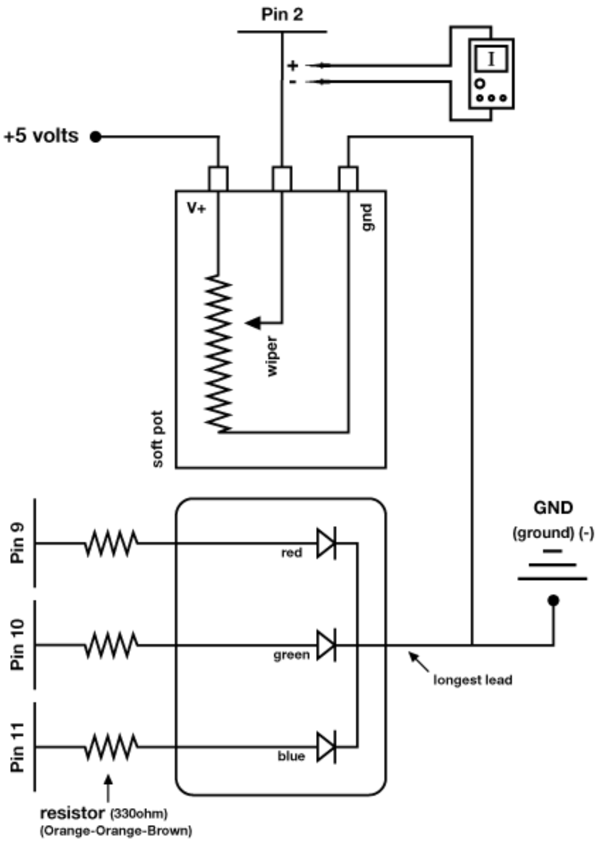
Got your soft potentiometer and RGB LED working? Great. Without looking at the code too much, mark on the soft potentiometer diagram below which areas cause which colors to be displayed.



Add the following code to your Arduino code.
In Setup: `Serial.begin(9600);`
In Loop after all other code:
`Serial.println(sensorValue);`
`delay(100);`
Now open the Serial Communication window.

This will let you see the values (slowed down a little with the delay line, to see real time output remove `delay(100);` as the soft potentiometer outputs them. What happens to the values after you stop touching the sensor? Explain why you think this happens. You may have to look up how this sensor works to figure this out.

Explain in your own words how you think you could use the soft potentiometer to turn the RGB LED on/off as well as controlling the color.



Calculate resistance of the potentiometer when it is blue, then green, then red. You will need to measure Voltage and Current, then calculate resistence because you can't see the RGB value while measuring resistance.

Blue:______ Ω Green:______ Ω Red:______Ω

SIK Worksheets v.1.0
Circuit 14, Middle School

Name:
Date:

Soft potentiometers come in a bunch of shapes and sizes. Explain how you can combine multiple soft potentiometers to create a basic touchscreen interface with X and Y position output...or just go build one instead of explaining how it might work. You've finished the SIK, what are you waiting for? In fact, forget these guiding questions, if you want just use this page to brainstorm what your next project will be.

SIK Worksheets v.1.0
Circuit 1, High School

Name:
Date:

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Did you get your LED (Light Emitting Diode) turned on? Great. Fill in the answers below using red LEDs.

Give values and units for Voltage, Current and Resistance for the whole LED circuit. Find Current by breaking the circuit. Calculate Resistance. Remember to include the resistor in your measurements.

Two LEDs in series, 5V: V =____ I =____ R =____

Two LEDs, parallel, 5V: V =____ I =____ R =____

One LED, 3.3V power: V =____ I =____ R =____

Two LEDs, series 3.3V: V =____ I =____ R =____

Two LED, parallel 3.3V: V =____ I =____ R =____

Replace your 330Ω resistor with a 10KΩ resistor.

Two LEDs in series, 5V: V =____ I =____ R =____

Two LEDs, series, 3.3V: V =____ I =____ R =____

What do you think would happen if you connected a 9V battery as your power source for the first circuit?

Assuming the same resistance as the original circuit, what would the current equal with a 9V power source? Show your work.

In the code below circle the "`setup( )`" method and explain below what it does in this instance.

Underline the code that turns the LED on.

```
 int ledPin =  9;

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

void loop()
{
  analogWrite(ledPin, 200);
  delay(1000);
  analogWrite(ledPin, 0);
  delay(1000);
}
```

Why does the code above use pin # 9 instead of pin # 0 or pin # 1? Explain why pin # 0 and pin # 1 are not options. Make sure you explain for both digital pins and analog input pins.

Explain why you might use LEDs on an illuminated shirt (or hat, etc) instead of other types of light bulbs.

SIK Worksheets v.1.0 **Name:**
Circuit 2, High School **Date:**

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Get your LEDs turned on? Great. Now you are going to add a dimmer switch to your LED on pin # 9.

What user interface component will you need to use as a dimmer?

Add the necessary text to the `oneOnAtATime( )` method for the code heavy way to add the dimmer.

There are three different way to add a dimmer without changing or adding code. Try to find one of these ways without destroying your LED.

Draw a schematic of your circuit in the space to the right, adding your dimmer component so that it works. There are four different ways to do this.

What other component does the dimmer component in this circuit act as?

The LED needs a PWM value that ranges from 0 – 255. The dimmer component gives you values from 0 – 1023. Write an equation below that will convert the value the dimmer component outputs to a LED friendly value.

Draw a logic flow chart of the LED with dimmer.

The LED values 0 – 255 actually represent 256 different values. Why is that?

Imagine your LED circuit (without dimmer) as a meter indicating a sensor reading. Decide what kind of sensor you would like to use as an input and describe in your own words what would cause the meter to rise and fall.

SIK Worksheets v.1.0
Circuit 3, High School

Name:
Date:

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Got your motor running? Upload Circ03Expansion Code to your Arduino. Fill in the answers below.

Give Voltage values for each PWM value listed at each multimeter location. You will need to change the code to measure the PWM values listed.

Position 1: PWM @ 100%, V = _______v

Position 1: PWM @ 75%, V = _______v

Position 2: PWM @ 100%, V = _______v

Position 2: PWM @ 75%, V = _______v

Position 3: PWM @ 100%, V = _______v

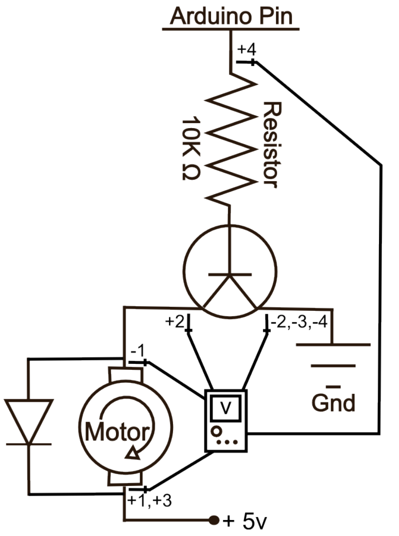
Position 3: PWM @ 75%, V = _______v

Position 4: PWM @ 100%, V = _______v

Position 4: PWM @ 75%, V = _______v

The amount the voltage decreases by as it passes through components is called voltage drop. What is the correlation between the various voltage drops you just measured?

In your own words explain what this transistor does and the ways in which the motor's action would change if it were hooked up directly to a 5V power source and a ground?



Without using any code how could you make the motor run the other direction?

A little history:

The word transistor is a combination of what two words?

What was the first type of transistor to be mass produced, and by who?

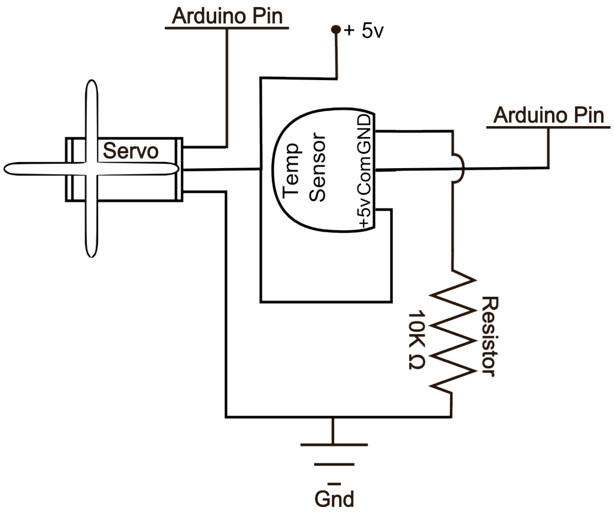
Transistors are used in almost ever piece of modern electronics and considered one of the most important inventions of the 20th century. What are some of your favorite items that contain a transistor? Name at least five.

SIK Worksheets v.1.0
Circuit 4, High School

Name:
Date:

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Got your servo running? Great. Upload Circ04Expansion to your Arduino and add a temperature sensor (pin 0) to your circuit so it controls the position of the servo depending on the temperature.



Decide what the parameters of your temperature gauge will be in Celsius. Find the line of code that controls this and change as necessary.

Using the `delay( );` command change the code so that the speed of the servo is also controlled by the temperature. You can make it move faster or slower depending on the temperature sensor input. Ideally you will only need to change a single line of code to do this. Write the line of code you used below.

Add a button to the schematic on the right which allows the user to change what aspect of the servo the temperature sensor controls. Extra credit if you can modify the code so your circuit does this in real life.

Define a servomechanism in your own words.

There are many types of servomechanisms that are not simple motors with position feedback, what is the most complicated servo you can think of?

Instead of a temperature sensor you could have added almost any sensor or interface component to your servo. Document at least three other options and explain briefly how you would control the servo.

It is possible to do amazing things with servomechanisms. In your own words, explain below how you could use the servo to create an autonomous marshmallow (because they are soft) launcher that corrects its angle depending on where the previous shot landed. Don't worry about how to get data about where the previous marshmallow landed, just explain how the servo would react to a marshmallow that went too far or too short.

SIK Worksheets v.1.0 **Name:**
Circuit 5, High School **Date:**

The byte in this circuit's shift register is used to turn LEDs on and off. It can also be used to represent many other types of data in binary. Explain how a number is written in binary, then write the number nine using ones and zeros.

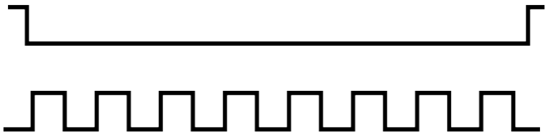
Does your shift register light up all the LEDs in a pattern? Great.

Explain why the shift register needs the clock pin as part of how it operates.

Fill in the diagram below if you were trying to shift in the following bits for data: 11100110

Latch:

Clock:

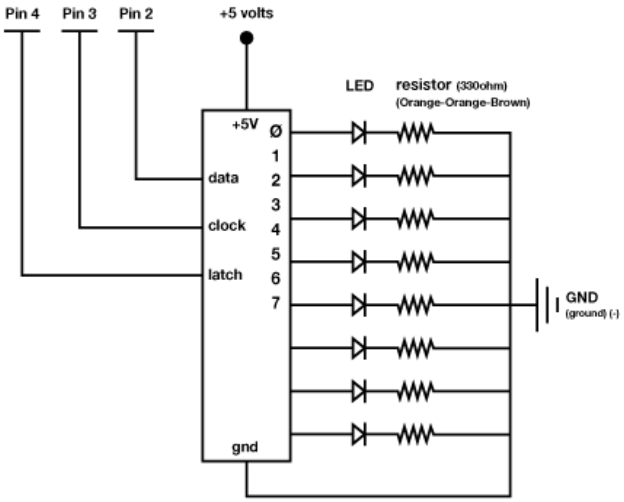


Data:

On the schematic to the right label the most significant bit's pin with MSB and the least significant bit's pin with LSB.
Assume the byte below is in your shift register chip.

Most significant bit Least significant bit
(MSB) ---> 1 0 1 1 0 1 0 0 <--- (LSB)

While shifting bits in and out of your chip you can move them so you drop either the MSB, or the LSB. Assuming you drop the LSB and are adding a 1 to the byte above, what is the resulting byte?



A little history:

What was one of the first known uses of the shift register?

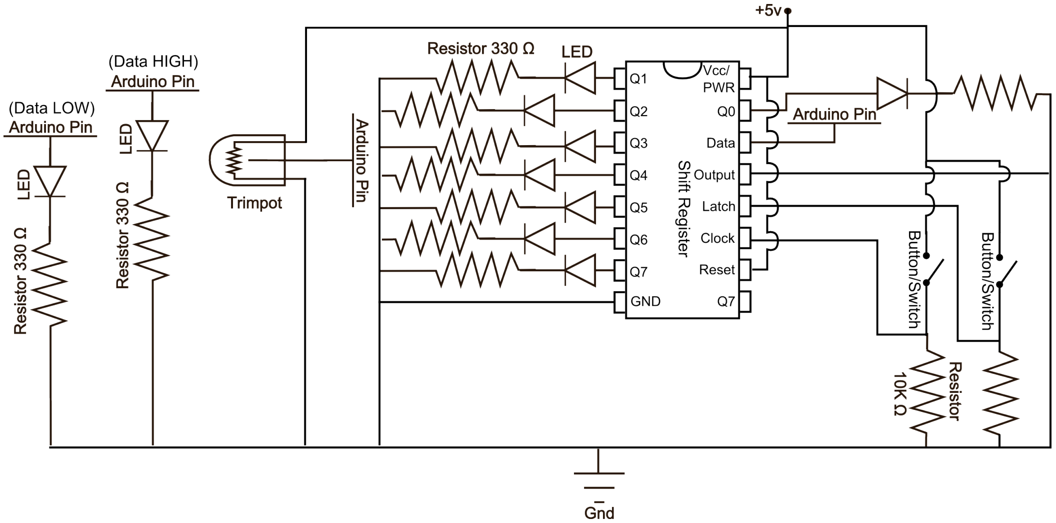
This shift register and LED combination is a powerful display or interface circuit. Explain a sensor or two you might attach to this circuit and what the LEDs attached to the shift register would signify. Example: connect a mic and each LED would represent a different musical note heard by the mic.

SIK Worksheets v.1.0 **Name:**

Circuit 5b, High School **Date:**

The byte in this circuit's shift register is used to turn LEDs on and off. It can also be used to represent many other types of data in binary. Explain how a number is written in binary, then write the number fifteen using ones and zeros.

Does your shift register light up all the LEDs in a pattern? Upload Circ05Expansion to your Arduino and add two buttons, two LEDs and a trimpot to your circuit, use the schematic below for reference.



Your two buttons now pulse the clock and latch the shift register. Make sure you don't confuse the two! You will use the trimpot to set your data either HIGH or LOW. Play with the trimpot to figure out which setting is HIGH and which is LOW. One of the indicator LEDs will light up depending on which value it represents. Then use the clock pulse button to send the data value to the shift register. To see the data that you have shifted into the register so far hit the latch button. To really get a feel for how shift registers work first set all the LEDs LOW, then start playing with different patterns of data values.

Before answering the questions below set all your pins all back to LOW, or off (Remember, LOW == 0)

Try setting just one data pin before hitting the latch button. What happens?

Try setting eight data pins before hitting the latch button. What happens?

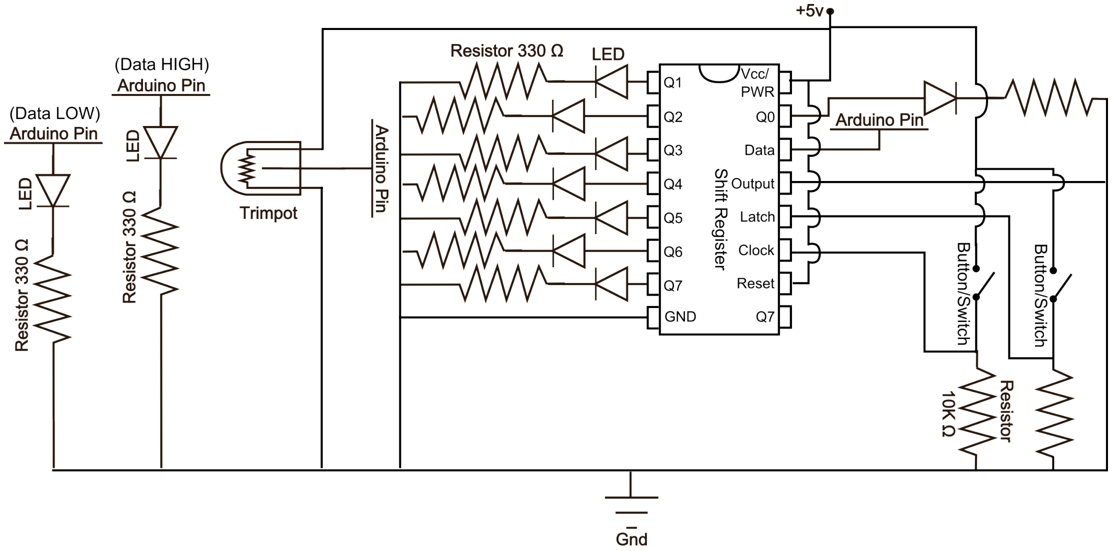
Now set seven data pins before hitting the latch button. What happens this time?

Now enter nine bits in the following order: 0,1,0,0,1,1,0,0,0. What does your LED pattern look like? Answer in binary.

Explain how the LED pattern and shift register would act if you were shifting out the Most Significant Bit instead of the Least Significant Bit. Find the one command or word you would need to change in the code to make this happen and write it below.

SIK Worksheets v.1.0 **Name:**
Circuit 5c, High School **Date:**

Got your shift register lighting up all the LEDs in a pattern? Upload Circ05Expansion to your Arduino and add two buttons, two LEDs and a trimpot to your circuit, use the schematic below for reference.



Your two buttons now pulse the clock and latch the shift register. You will use the trimpot to set your data either HIGH or LOW. Play with the trimpot to figure out which setting is HIGH and which setting is LOW. One of the indicator LEDs will light up depending on the value. Then use the clock pulse button to send the data value to the shift register. To see the data that you have shifted into the register so far hit the latch button.

Before answering the questions below set your data pins all back to LOW, or off (Remember, LOW == 0)

Which circuit, original or expanded makes more sense to you?________________________________

Using this interface you have more control over the shift register than the original code. Explain the difference between this circuit and the original. Use examples from the original code and explain what physical element has replaced it. Also explain what the various states of the components are.

Because you decide when to “latch” your data in you can control all the pins in any order you like as long as you “clock” in the proper sequence. This lets you decide which pins are ON or OFF without having to cycle through them all. Decide on eight different circuits (or objects) you would like to turn on and off using a register and then explain at least two different patterns you would send the shift register to control these objects. Use binary to write the patterns. Example: servo, servo, egg beater motor, spray on butter object, servo, waffle iron, servo, hot plate. 1111100: first two servos pour ingredients, egg beater mixes, butter sprays on waffle iron which is heating up, servo and hot plate off. 00000111: first two servos reset, egg beater off, no butter, waffle iron stays on, servo tilts waffle off of iron onto hot plate which keeps waffle warm. Note: Zero does not always mean off, it can make the circuit (or object) do something else, like reset a servo position or squirt syrup instead of butter.

SIK Worksheets v.1.0
Circuit 6, High School

Name:
Date:

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Got your incredibly annoying song blaring out of your tiny speaker? Great.

The piezo speaker uses digital pulsing (on or off) to create an analog sound value which can rise or fall in an analog fashion even though it is technically digital. What other digitally simulated analog signal is this similar to?

Although the action of the piezo speaker is similar to the simulated analog signal, what word or command in the code shows us that it is different? What command would you use if you wanted to use the simulated analog signal instead of the purely digital one?

Is it possible to make the piezo speaker play a note so low that the human ear cannot hear it?

Piezo elements are used for many things other than playing music. In fact you might have a piezo element in your pocket right now. List at least three usages of a piezo element other than a piezoelectric speaker.

Find and correct the three errors in the code below.

```
void loop() {
for (int i = 0; i < length; i++) {
       if  notes[i] == ' '  {
         delay(beats[i] * tempo);
// rest
       } else {
playNote(notes[i], beats[i] *
tempo);
       }
          pause between notes
       Delay(tempo / 2);
     }
   }
```

Underline all instances of matrices in the code above.

Fun fact:
There are now digital turntables which manipulate digital sound samples similar to this piezo element. When the sample is slowed down the sample's frequency drops as well due to the increase in gap size between the digital values. When the sample is sped up the pitch rises because the gaps decrease.

There are many different outputs you can couple with this circuit. For example: add a servo with a backdrop to indicate which note is being played, or couple it with an RGB LED that shines a different color depending on where in the scale the note is positioned. Write, in plain English, how you would control a chosen output as well as the piezo component.

SIK Worksheets v.1.0 **Name:**
Circuit 6, High School **Date:**

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Got your incredibly annoying song blaring out of your tiny speaker? Great.

Now you are going to add volume control to your piezo speaker. First place a 330Ω resistor on the circuit. Draw the two possible schematics of this new circuit to the right. What changes when you add the 330Ω resistor?

Next replace the 330Ω resistor with a 10KΩ resistor. What changes this time? What does this lead you to believe about resistors and the piezo speaker? Explain.

Now replace the resistor with the potentiometer. Measure the resistance of the potentiometer and write below the lowest resistance value it can be set to and the highest resistance value it can be set to.

Lowest resistance: _______________Ω Highest resistance: _______________Ω

But wait! Can't we just turn down the volume using a lower PWM value in the code? Why are we changing resistors when it's so much easier to just rewrite the code a little? Find the line of code you will need to change to try using PWM to control the volume instead of a resistor and write it below.

Now actually change the code and listen to the results. Can't hear any difference? Try using a lower PWM value. You should definitely notice a difference now. That's different from changing the volume right? Now use your potentiometer to change the PWM value of the piezo speaker circuit. You will need to change the code to do this. Write the three essential lines of code you used to make this happen below, don't forget semicolons. (Hint, one of them is a variable declaration before the `setup()` method.)

Does your piezo speaker turn off when you turn your potentiometer all the way one way? This is because the analog values go up to 1023 but the PWM only go to 255. You can use the `map()` method to fix this. Write the line of code which will fix the problem below. (If it still turns off with map, make sure your PWM value never goes all the way down to zero.)

The effect that changing the PWM has on “Twinkle, Twinkle Little Star” is kind of like an effect that many musicians use in on their instruments in modern music. What is that effect?

SIK Worksheets v.1.0
Circuit 7, High School

Name:
Date:

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Can you turn your LED on and off using both buttons? Great. Upload Circ07Expansion Code to your Arduino and add an RGB LED to pins 9, 10 and 11. Check the code if you are unsure which leads go to which pins.

The buttons in your circuit now adjust the variable “RGBValue” either up or down. What are the upper and lower parameters of “RGBValue”?

With the code as is, what happens if you press the “down” button while pressing the “up” button? Why do you think this is?

What could you add to the code to fix this bug?

In the code below underline the command that happens when the button is not being pressed.

```
void loop()
{
  buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
}
```

Explain the difference between = and ==.

Draw a Logic Flow Chart of the expanded circuit here:

In the space below draw the symbols for a two way switch (SPST), a three way switch (SPDT), and a double pole switch (DPST). Label all three.

Buttons are everywhere, however it is possible to substitute other user interface components for buttons, list at least three components that you could switch with a button in some way.

SIK Worksheets v.1.0
Circuit 8, High School

Name:
Date:

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Can you turn your LED up and down using the potentiometer? Potentiometers are also called trimpots.

Calculate percentage for each of the analogWrite values, then draw a line from the PWM code on the left to the corresponding PWM diagram on the right.

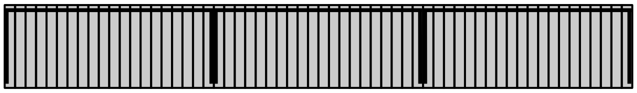
analogWrite (ledPin, 0); _______%

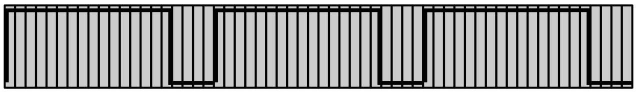
analogWrite (ledPin, 200); _______%

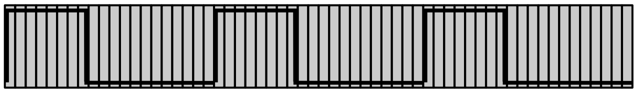
analogWrite (ledPin, 255); _______%

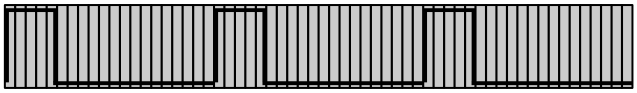
analogWrite (ledPin, 70); _______%

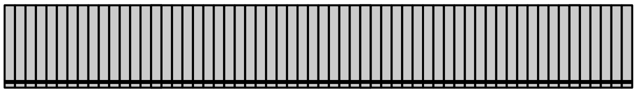
analogWrite (ledPin, 100); _______%



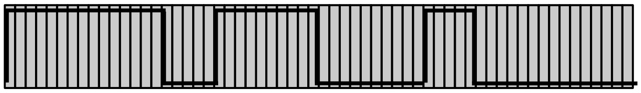








Write the equation you used to figure out these %.



Describe how the potentiometer is being adjusted according to the PWM diagram above.

Who invented the potentiometer and when?

What basic component does a potentiometer act like when it is not being adjusted?

In your own words describe what voltage dividers do.

Describe how you would use potentiometers to control a marshmallow (because they are soft) launcher's trajectory. What other pieces of hardware would you need to create this marshmallow launcher?

SIK Worksheets v.1.0
Circuit 9, High School

Name:
Date:

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Can you turn your LED up and down using the photoresistor? Great.

In the code you uploaded to your Arduino board change the line:
`lightLevel = map(lightLevel, 0, 900, 0, 255);`
to: `lightLevel = map(lightLevel, 0, 900, 255, 0);`
How does this change the way your circuit acts?

Leave the code above in and turn your photoresistor so that it faces the LED. Turn the lights off. Does your LED turn all the way off? Why is this?

What two lines do you need to add to your code to see what the output values from the photoresistor are and where do you need to add them?

There are three reasons the code below will not work, find all three errors and change or add the necessary code so it does work.

```
int lightPin = 0;
int ledPin = 8;

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

void loop()

 int lightLevel =
analogRead(lightPin);
 lightLevel = map(lightLevel, 0,
900, 0, 255);
 lightLevel =
constrain(lightLevel, 0, 255);
 analogWrite(ledPin,
lightLevel);
}
```

Who invented the photoresistor, or photocell, and where was it invented?

Photoresistors occur in nature. List examples you can think of below. (You are using at least one right now.)

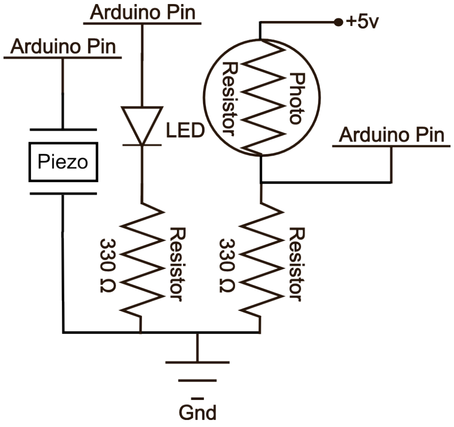
From the code above copy the command you would need to change if you wanted the LED to light up only when the photoresistor value is above 50%.

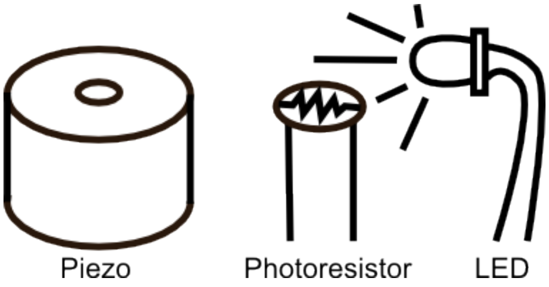
Write below what you would need to change the command to so that it functions as described above.

Photoresistors are great for light control, what else would you like to control with them? You can turn other circuits on or off by turning on your lights or opening your blinds. List at least three circuits.

SIK Worksheets v.1.0 **Name:**
Circuit 9, High School **Date:**

Now you are going to use your SIK to create an alarm system using analog and PWM signals. Upload the Circ9Expansion Code onto your Arduino and setup your breadboard according to the schematics below.





This simple alarm is designed so that when something comes between the light source (LED) and the photoresistor the piezo element starts it's annoying beeping. If you open your Serial communicator you will see it printing out a PWM value for the strength of the LED and an analog value for what the photoresistor sensor is receiving in the way of light. Without changing any of the code your alarm should work if you turn off the lights and pass your finger between the LED and the sensor.

What changes in this circuit and causes the alarm to go off? It's not just a lack of sunlight. Explain what is happening with the electricity. Write below and explain the exact variable that changes as well as the value that causes the alarm to be triggered.

In order to make the alarm work during daylight you will need to be out of any direct lighting and you will need to change one of the values in the code. In the space below, write the line you changed in the code and explain why.

What component in the SIK do you think you could use to physically change the sensitivity of the photoresistor so you don't have to change the code whenever the sunlight levels change? Explain how.

SIK Worksheets v.1.0 **Name:**
Circuit 10, High School **Date:**

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

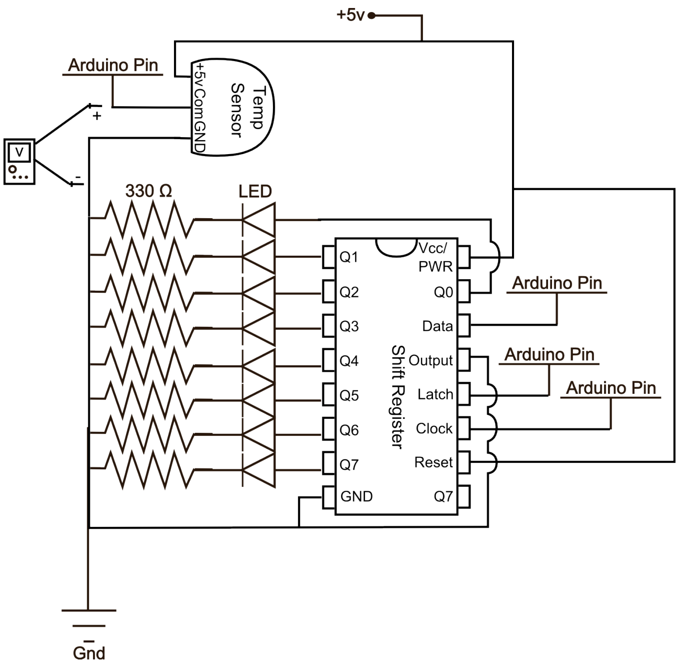
Does your temperature sensor work? Great. Upload Circ10BExpansion Code to your Arduino. Attach the shift register and LEDs as shown in the schematic below. You may have to alter some code depending on how hot or cold it is where you are.

Write voltage values for the two temperature values below. Also record the amount of LEDs that light up with each temperature. For the first temp use whatever temp your room currently is.

_____°F ≈ _____°C (room temperature, fill in degrees)
V =______v LEDs =___

98.6°F = 37°C (use a cup of cocoa to warm up sensor)
V =______v LEDs =___

50°F = 10°C (use an ice pack to cool down sensor)
V =______v LEDs =___



Using the values above, formulate an equation so you can calculate values for the temperatures below.

105°F ≈ 40.5°C V =______v LEDs =___
86°F ≈ 30°C V =______v LEDs =___
32°F ≈ 0°C V =______v LEDs =___
-49°F ≈ -45°C V =______v LEDs =___

Write the equation below that solves for the change in voltage (V) given a change in temp (X).

It is impossible to get a reading for one of the temperatures above. Place an X beside this value.

Draw a Logic Flow Chart of the circuit here:

There are four main types of temperature sensors, thermocouples, resistance temperature detectors, thermistors, and temperature-transducing ICs. Which one are you playing with and is it analog or digital?

How is this similar to a LED? To a motor?

Find a way to quickly change the temp read by the sensor. How fast can you get it to change by 20°F?

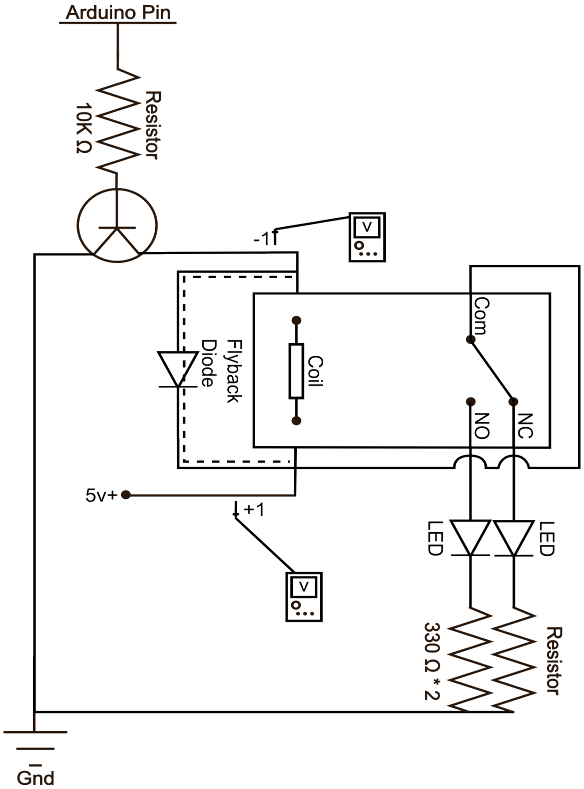
SIK Worksheets v.1.0
Circuit 11, High School

Name:
Date:

How is this circuit, or a circuit like it, used in everyday life? Provide at least three examples.

Does your relay work? Great.

There are many different types of relays. List at least three and explain the differences between them.



Give values for Voltage, Current and Resistance for the multimeter position shown. Break the circuit or use Ohm's law to solve for current and resistance. You should get two different sets of values depending on the action of the relay.

Meter 1:
Value # 1: V =______ v I =______ mA R =_____ Ω
Value # 2: V =______ v I =______ mA R =_____ Ω

Draw arrows on the dotted line to show possible direction of current flow when Arduino is turned off.

Explain the two sets of values for the Voltage, Resistance and Current and what each set does.

Explain how a diode effects the current flow of a circuit.

Given what you answered above, explain what you think is the reason for the Flyback Diode, also explain what might happen without this Flyback Diode.

What machine houses your favorite relay? Why is it your favorite relay? Because of the machine that houses it? Because it keeps someone safe? Because it's really big and powerful? Explain.

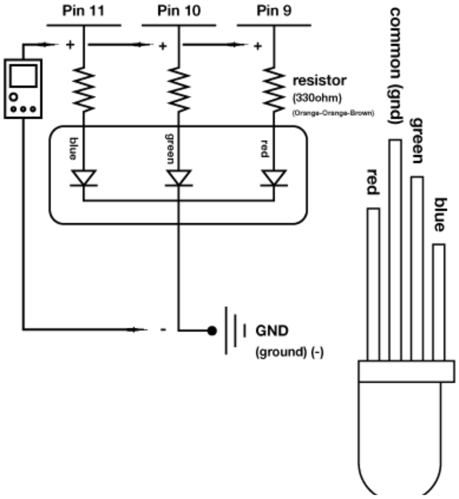
SIK Worksheets v.1.0 **Name:**

Circuit 12, High School **Date:**

This LED has three inputs and three outputs. What other circuits have three inputs? Three outputs? Provide an example of each.

Does the RGB LED work? Great. Hint for questions below: if you're really stuck add a button to your circuit that uses `Serial.println();` to print RGB values when you press it.

Connect a multimeter to each line that is connected to an Arduino pin. Notice how the voltage changes while you use the sensor or interface coupled with each pin.



What should the voltage values for each pin be to make the RGB LED as red as it can get?
Pin 11 V =____v Pin 10 V =____v Pin 9 V =____v

What should the voltage values for each pin be to make the RGB LED as yellow as it can get?
Pin 11 V =____v Pin 10 V =____v Pin 9 V =____v

What should the voltage values for each pin be to make the RGB LED as white as it can get, but at 1/2 intensity?
Pin 11 V =____v Pin 10 V =____v Pin 9 V =____v

Given that there are 256 (0 – 255) different intensity values for each color in an RGB LED, how many different colors can an RGB LED emit? Show your work.

How many different colors can a RGBY LED emit?

Why are there 256 LED intensities, and not 300?

Light is a small part of a big group of electromagnetic fields called the Electromagnetic Spectrum. Name at least three other electromagnetic fields and state if their frequencies are higher or lower than light.

Who discovered the theory of Electromagnetic Spectrum? Who expanded it beyond light?

Upload Circ12BExpansion to your Arduino and add a piezo element to digital pin # 1. Change the code so you don't go crazy listening to “Twinkle, Twinkle Little Star”. Explain how you could add another RGB LED so it also changes color depending on the note your piezo plays, but with different code so it displays a different color.

Other than Red, Green and Blue, what else could RGB stand for? Example: Roasted Gooey Baloney, Random Gargantuan Baboons, Rappers Got Beef, or Robot Guild Butlers. List at least three.

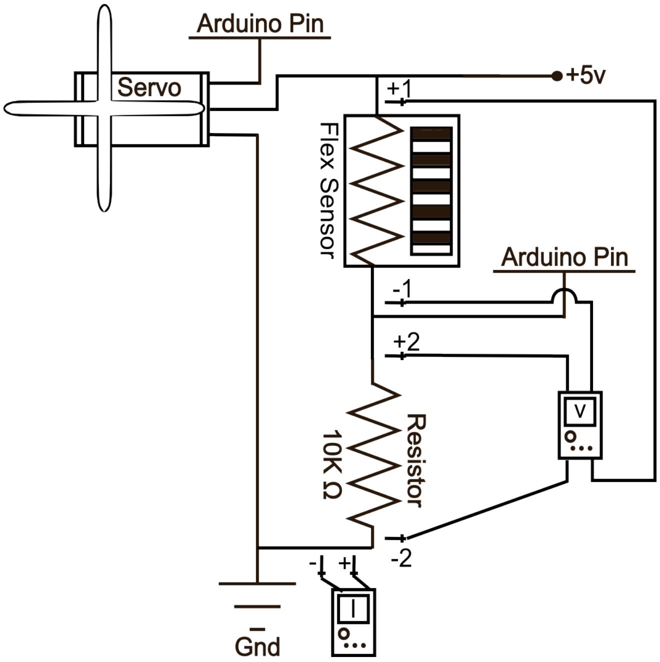
SIK Worksheets v.1.0 **Name:**
Circuit 13, High School **Date:**

Now we're starting to work with some more complicated sensors. The flex sensor has tons of real world applications. List three and explain why you can't use a regular potentiometer instead of a flex sensor. Example: use the sensor to measure the flex on a fishing pole and cut the line if the pole ever comes close to breaking. You could not use a potentiometer because it would be difficult to attach it.

Got your flex sensor and servo working? Great.

Give two values for Voltage, Current and Resistance for each multimeter placement. The first value is without bending the flex sensor and the second is with the flex sensor bent so the sensor creates a half circle. Don't crimp the flex sensor, just bend it. Find Current by breaking the circuit and using the multimeter. Calculate resistance.

Multimeter 1, no bend:
V = ______v I = ______mA R = ______Ω
Multimeter 1, with bend:
V = ______v I = ______mA R = ______Ω
Multimeter 2, no bend:
V = ______v I = ______mA R = ______Ω
Multimeter 2, with bend:
V = ______v I = ______mA R = ______Ω



For each multimeter position mark two Xs in additional places where you could attach the multimeter (+ and -) to get these same readings.

Using a protractor to measure the servo angle, and the Serial Monitor to output the analogRead value of the flex sensor. Create a graph that shows correlation. Remember to label your graph.

There are many kinds of flex sensors, list 3.

What kind of flex sensor are you working with now?

What other kinds of sensors is this similar to?

How could you figure out the values for the second multimeter placement given the first set of values?

If the flex sensor itself is a resistor, why is the 10K resistor necessary? Explain.

Take your favorite hypothetical project that you have written about so far in these worksheets, explain how and why you might add at least one flex sensor to this project.

SIK Worksheets v.1.0 **Name:**
Circuit 14, High School **Date:**

The soft potentiometer is very touchy; sometimes you will notice incorrect readings due to how you touch the sensor. Be careful not to touch below the sensor pad, you will short out the sensor. Also the sensor reads a value even when you are not touching it. Explain how this alters the ways in which you can use this sensor. Explain at least one possible fix or work around.

Add the following code to your Arduino code.
In setup: `Serial.begin(9600);`
In loop (at end):

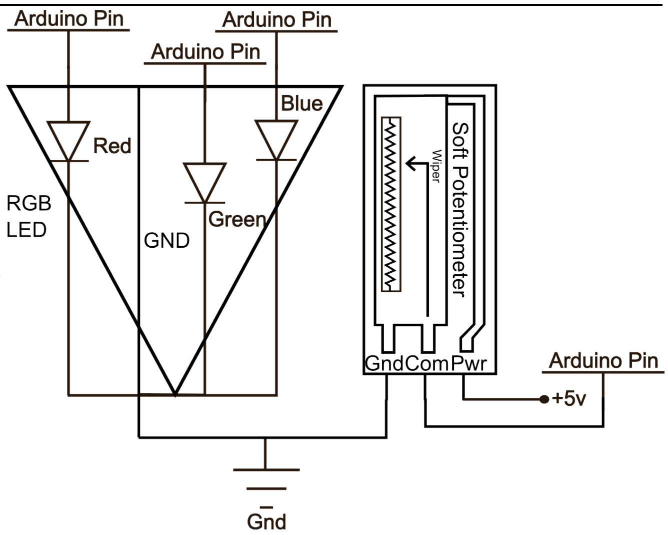
```
  Serial.println(sensorValue);
  delay(100);
```

Now open the Serial Communication window. This will let you see the values (slowed down a little with the delay line, to see real time output remove `delay(100);`) as the soft trimpot outputs them. What happens to the values after you stop touching the sensor? Explain why you think this happens.

Give values for sensorValue, Voltage, Current and Resistance for each question. To measure resistance; disconnect trimpot, press area that corresponds to color and attach the multimeter to the com line and ground. Find Current by calculating with Ohm's Law.

RGB in red range: V =____v I =____mA R =____Ω
sensorValue = ______
in blue range: V =____v I =____mA R =____Ω
sensorValue = ______
in green range: V =____v I =____mA R =____Ω
sensorValue = ______
in yellow range: V =____v I =____mA R =____Ω
sensorValue = ______

What RGB values do you need to display purple?



Touch the sensor lightly (don't push or hold it) and run your finger from one end to the other. What happens? Why do you think this is? Explain.

You can make the RGB LED display red when you touch the bottom and blue when you touch the top by changing two lines of code. Write one these lines of code below as well as what you need to change it to.

Write an if statement you could add to your `loop()` method that causes the RGB LED to display purple when you touch the absolute top of the trimpot.

Imagine your soft potentiometer is thirty feet long. Explain at least three things you could use it for.

SIK Worksheets v.1.0
Circuit 14:

Name:
Date:

Soft potentiometers come in a bunch of shapes and sizes. Explain how you could combine two soft potentiometers, two servos and a slingshot to create a machine that launches marshmallows (because they are soft) towards wherever the two soft potentiometers tell it to, include a schematic or logic flow chart in the blank section .... or just go build one instead of explaining how it might work. You've finished the SIK, what are you waiting for? In fact, forget these guiding questions, if you want just use this page to brainstorm what your next project will be.

SIK Worksheets v.1.0 **Name:**
PWM **Date:**

We know by now that PWM is represented in Arduino language as a value somewhere between 0 and 255. We also know that the PWM signal is just the Arduino turning the PWM pin ON, or HIGH, then LOW, or OFF, a bunch of times really, really fast. The thing is the computer can turn the signal HIGH and LOW a lot faster than a human can follow. So what does the PWM signal look like to us and how can we measure it? Good question. Upload the PWMExpansion Code to your Arduino, attach PWM pin # 9 to ground on your breadboard and measure the voltage between the two wires. Switch values and reload the code as necessary to fill in the table below.

Fill in the table below.

Percentage:	0	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
PWM value:	0					127					255

What is the equation you use to convert the percentage to PWM values and back again?

____

____

Why does the PWM value only go up to 255 if it represents 256 different values?

____

Often you will use analog sensors with your digital microprocessor. These analog sensors are even more difficult than the PWM signal. They talk to the microprocessor in 1024 little pieces! Fill in the table below.

Percentage:	0	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Analog value:	0					511					1023

What is the equation you use to convert the percentage to analog values and back again?

____

Now fill in the table below, it should be pretty easy by now.

Percentage:	0%	25%	50%	75%	100%
PWM value:	0				255
Analog value:	0				1023

What is the relationship between the analog and PWM value? Explain with words or math.

____

____

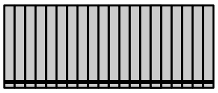
What is one advantage to representing an analog sensor's value with 1024 numbers instead of just 100?

____

____

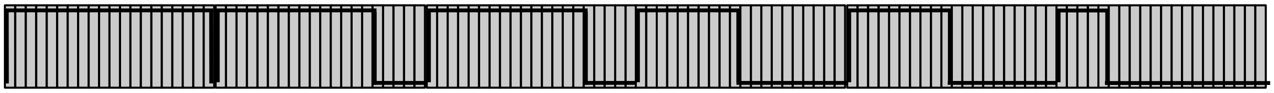
SIK Worksheets v.1.0 **Name:**
PWM **Date:**

Below are a bunch of different PWM wave diagrams. The “window” is this size: Record below each wave diagram the starting value in the unit given, the highest value the PWM wave reaches (PWM value, 0 - 255), the lowest value the PWM wave reaches (PWM value, 0 - 255), and the ending value of the wave diagram. Finally, draw a line through the diagram showing the PWM value as it rises and falls.



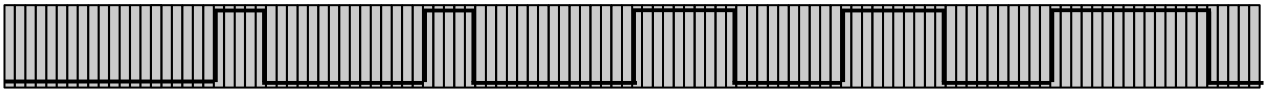
Remember for the SIK PWM values: 100% = 255 PWM = 1.8mV

1.



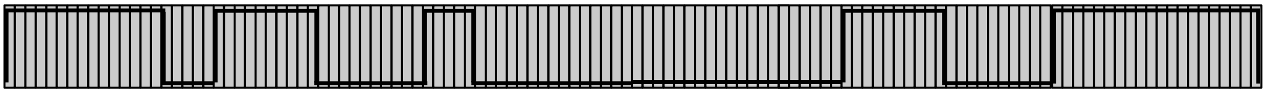
_______starting PWM ______Highest PWM _______Lowest PWM _______ending PWM

2.



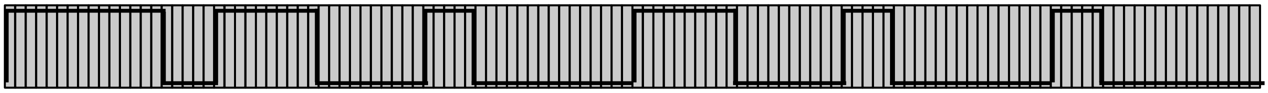
_______starting % ______Highest PWM _______Lowest PWM _______ending %

3.



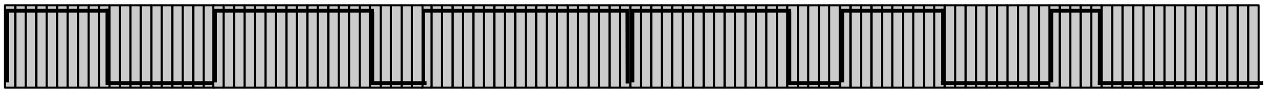
_____starting voltage ______Highest PWM _______Lowest PWM ______ending voltage

4.



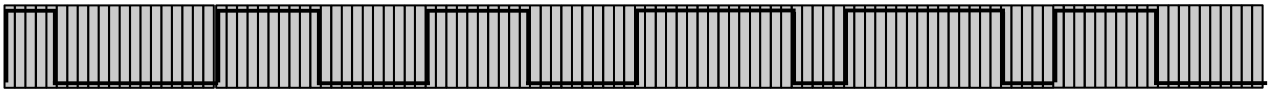
_______starting PWM ______Highest PWM _______Lowest PWM _______ending PWM

5.



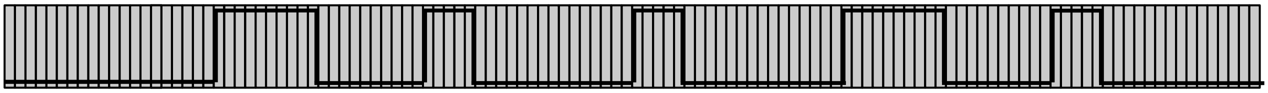
_______starting % ______Highest PWM _______Lowest PWM _______ending %

6.



_____starting voltage ______Highest PWM _______Lowest PWM ______ending voltage

7.



_______starting PWM ______Highest PWM _______Lowest PWM _______ending PWM

These PWM windows are nicely divided into ten sections so you can count out the windows and figure out the PWM values and percentages easier. Real PWM signals are not so convenient and are usually represented by one of the following with no diagram: the PWM value, a percentage, or a voltage value.

SIK Worksheets v.1.0 **Name:**
PWM **Date:**

We know by now that PWM is represented in Arduino language as a value somewhere between 0 and 255. We also know that the PWM signal is just the Arduino turning the PWM pin ON, or HIGH, then LOW, or OFF, a bunch of times really, really fast. The thing is the computer can turn the signal HIGH and LOW a lot faster than a human can follow. So what does the PWM signal look like to us and how can we measure it? Good question. Upload the PWMExpansion Code to your Arduino, attach PWM pin # 9 to ground on your breadboard and measure the voltage between the two wires. Switch values and reload the code as necessary to fill in the table below.

Percentage:		0%	10%	25%	33%	50%	66%	75%	80%	90%	100%
PWM value:		0		63.75				191.25			
Analog Value		0				511.5					1023
Voltage (mV)		0									

One of the easiest ways to find a correlation between numbers is to look at a lower set of values and compare them to a larger set of values with a common denominator. Look at all the values for 10% and then look at all the values for 100%. If there is a pattern it should be easier to see it with these two set of values simply because they are different by a factor of 10, or one decimal place.

Using the table above create an equation that you can use to change PWM or Percentage into an analog value. Don't forget that zero is a value in PWM and analog.

____=analog value

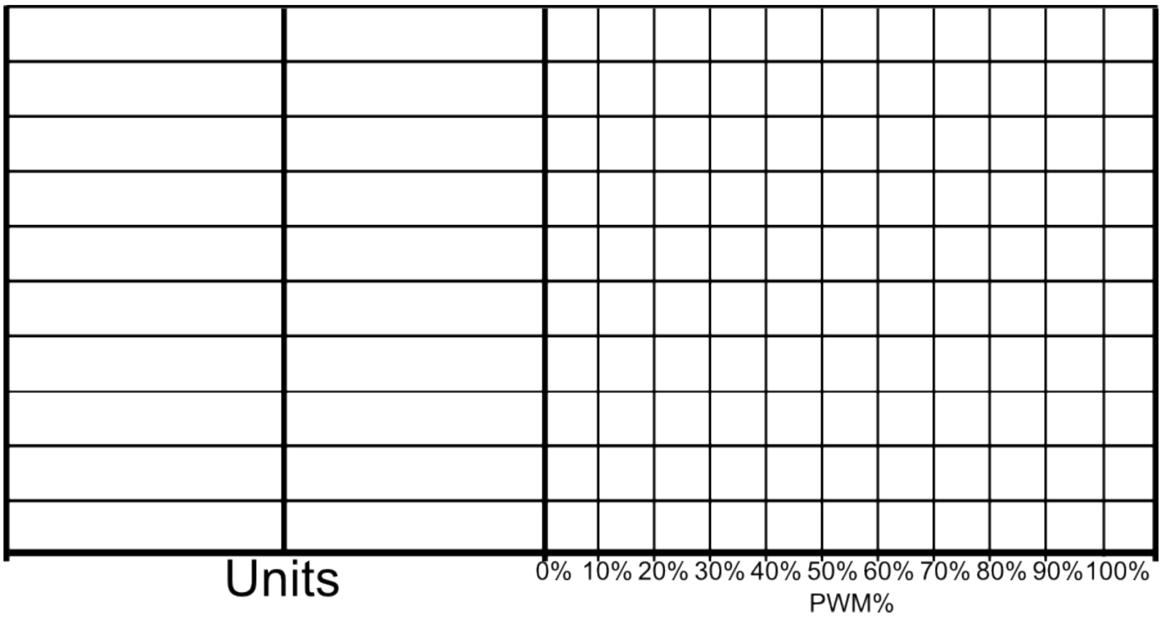
Using the table above create an equation that you can use to change a PWM or Percentage value into a voltage value. Take into account any difference in units that may make the math weird.

____= voltage (mV)

Now, you should be able to mash these two equation together into one equation that you can use to convert from analog to voltage. Think this is tough? Lucky we're not using a temperature sensor and making you also convert Fahrenheit and Celsius and resistance values! Write your equation below.

____

Using your equations fill in the line graph below for the analog and voltage variables above. Label your graph wherever you think necessary. Use a different color for the analog and voltage lines.



SIK Worksheets v.1.0
PWM

Name:
Date:

Write below these different values whether they are a PWM value, an analog sensor value, a percentage or a voltage. Then match each value below with the PWM wave diagram that best represents them by drawing a line from the value to the diagram.

50% ________________

1.8 mV ________________

analogWrite(pin, 64) ________________

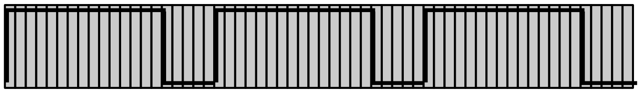
analogRead(pin)= 768 ________________

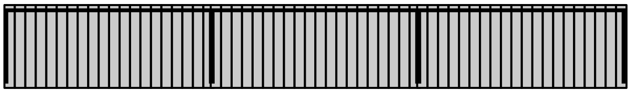
511 ________________

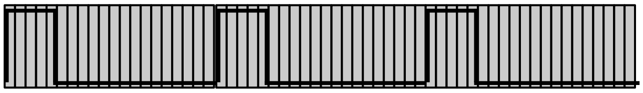
194 out of 256 ________________

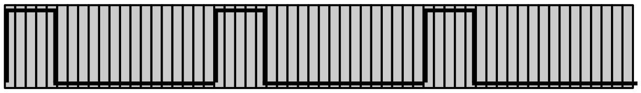
.45mV ________________

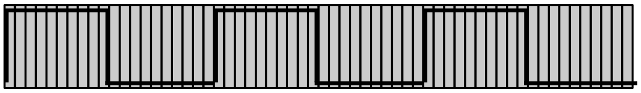
100 out of 100 ________________

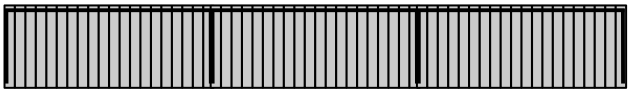


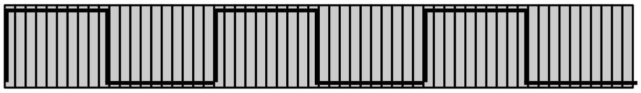


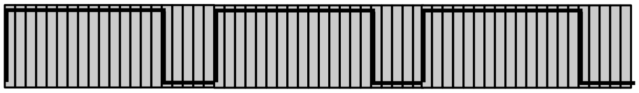












SIK Worksheets v.1.0
Fritzing Version 0.4.3, What is it?

Prototype to Product

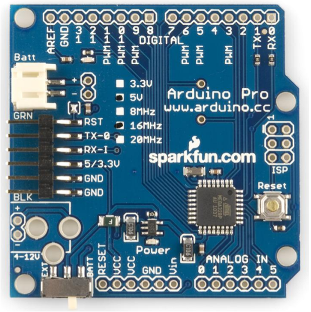
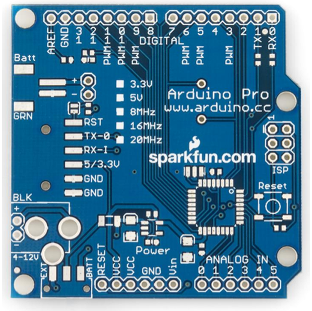
Ok, maybe you've created some sweet circuits using the S.I.K. and now you're wondering what is next. How do you turn your mass of components, wires and breadboard into something you can solder components onto and put into a tidy little package? Or in some cases a gigantic, take over the world, robot. If this describes you then you are looking to step up to the world of of virtual prototyping and printed circuit boards. Uh... what a second, what exactly does that last sentence mean?

Virtual Prototyping

You know all those circuits you have been putting together using your breadboard? Virtual prototyping is the process of recreating those electrical circuits in a computer application so you can turn them into a finished product. There are many different applications you can use for virtual prototyping, they range from fairly simple to very complicated. All of these applications create things called “Gerber files” which are the plans, or layouts, that an inventor can send to a printed circuit board manufacturer so the manufacturer can create the actual printed circuit board. This section explains how to use an introductory application called Fritzing.

Printed Circuit Boards (PCBs)

Printed circuit boards are the boards inside of most electrical items. You have probably already seen them (they are everywhere). Printed circuit boards are also referred to as PCBs. PCBs are the second to last step of creating your invention. They are basically boards with electrical paths inside and on top for hooking up all the necessary components.

 Populated Arduino Pro PCB (Parts on it)	 Unpopulated Arduino Pro PCB (Parts not on it)

Fritzing

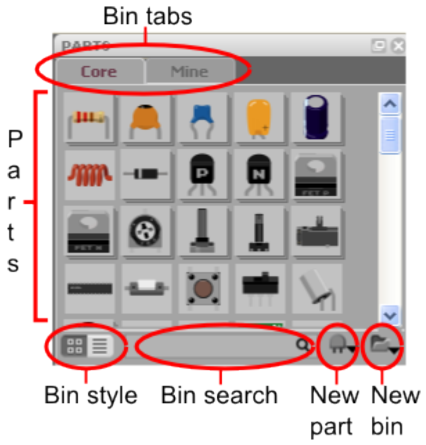
Fritzing is a free, open source application with an established online community. Fritzing can be used to create single sided PCB layouts which you can send to PCB manufacturers for mass production. This is a big step so it is important to double and triple-check everything about your design before, during and after this process. You don't want to spend money ordering a bunch of PCBs only to find out that you overlooked a detail and your final product won't work like your prototype.

In order to properly use Fritzing you need to understand the three different views. The breadboard view is for recreating your own physical breadboard and components, the schematic view simplifies the connections and components for easier viewing and the PCB layout view allows you to place the actual leads and pins for connecting components.

In order to help you create PCBs Fritzing outputs Gerber files and an etchable .PDF file. There are eight different types of Gerber files but Fritzing creates six because it only uses one side of the PCB. Gerber files and exporting are addressed at the end of this section.

Fritzing Parts Menu

The Parts menu is where you will find all the components you will need to create your virtual prototype. The standards components, wires, PCB parts and even a ruler are already there, ready for use under the “Core” tab. The rest of the buttons in this menu are outlined below.



Bin tabs: Bins are lists of parts that are available for use in Fritzing, for your convenience you can create your own bins to hold only the parts you need for a particular project. If you wish to create your own “bin” of parts there is already another tab labeled “Mine” which you can use. Simply click on the tab labeled “Mine” to see the parts in this bin, drag and drop from the core menu to add parts or use the new part sub menu. Right click on a part to activate the part editor window for modifying it for your particular purpose.

Parts: This is where your parts are listed. The images give you an idea of what each part is. Place your cursor on a part to see the name and properties in the Inspector Menu. Right click on a part to edit, export or remove. The edit option will pop-up a part editor window. You can edit the image, connections and properties of a part in this window. For more on this see Fritzing Part Creation.

Bin style: This button allows you to switch between two viewing styles. Try it out, see which you like.

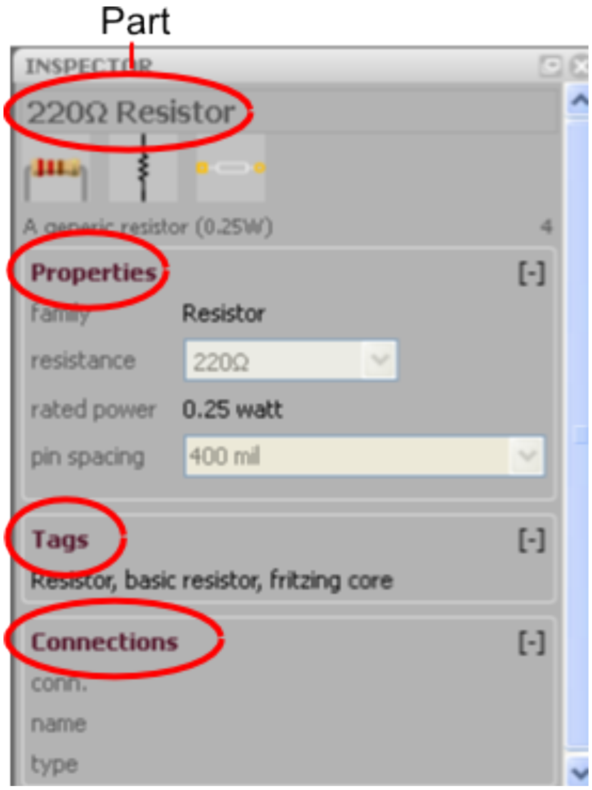
Bin search: Enter the part or information about a part to find it. For example: to find a resistor with a value of 220Ω you can enter either “resistor” or “220”, results are displayed in the “search” bin tab.

New part: This sub menu allows you to create or import a new part, and edit, remove or export an existing part. The new and edit options will pop-up a part editor window. You can edit the image, connections and properties of a part in this window. For more on this see the Fritzing, New Part Creation section.

New bin: This sub menu allows you to control the bin tabs. You can create new bins, as well as open, save, export, close and rename existing bins.

Fritzing Inspector Menu

The Inspector menu is where information about the currently selected part is displayed.



220Ω Resistor

A generic resistor (0.25W)

Properties [-]

family	Resistor
resistance	220Ω
rated power	0.25 watt
pin spacing	400 mil

Tags [-]

Resistor, basic resistor, fritzing core

Connections [-]

conn.
name
type

Part name and images: Pretty self explanatory, this is where information and images of the part you have selected are displayed. The three different images from left to right are the breadboard, schematic and PCB silkscreen images.

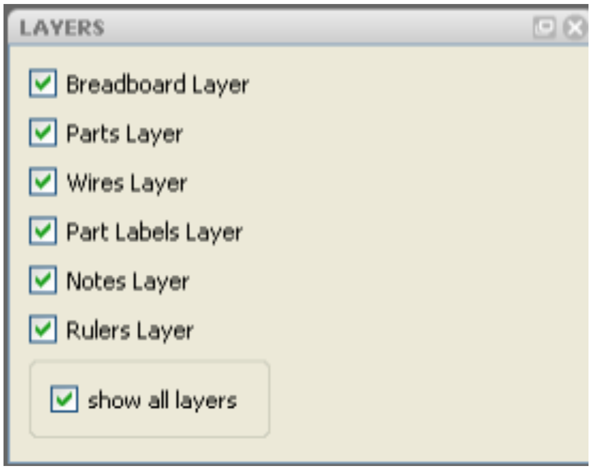
Properties: This sub menu displays properties particular to the part selected. The family section displays the part's component type. Below the family section various properties of the part are displayed. In this example resistance is an electrical value that you can change to fit your project's needs. Rated power is the amount of power this resistor can safely tolerate. Pin spacing indicates the amount of space between the two pins on this part, at this point in development pin spacing only effects the PCB footprint so you will not see any difference in the other views. Other examples of properties you may see in this sub menu include: Capacitance, rated Voltage, component type (crystal or ceramic for example), package type, doping (impurity type of transistor), maximum resistance and more depending on the part!

Tags: This portion is in development, for now it includes information that designates where a part is located and text that can be used to search for the part.

Connections: Once you have placed a part in your virtual prototype the connections made with this part will show up in this menu. Place your cursor over the wires (AKA leads) on the part in the main view to see the connection information. The Conn. field displays how many items are connected to to the lead. The name and type fields display the name of the connection (pin1, pin0, wire1, wire2, etc.) and the type (wire, male, female, etc.).

Fritzing Layer Menu

The Layers menu is where you can turn the various layers of the view on and off. Some of these layers are specific to the view, but others are available in all views.



☑ Breadboard Layer
☑ Parts Layer
☑ Wires Layer
☑ Part Labels Layer
☑ Notes Layer
☑ Rulers Layer
☑ show all layers

Breadboard Layer: Shows the breadboard, available only in breadboard view.

Parts Layer: Shows parts, available in all views.

Wires Layer: Shows wires, available in all views.

Part Labels Layer: In order to show the parts labels of a particular part right click on the part and select “show part label”, shows all parts labels, available in all views.

Nets Layer: Shows connecting wires that are not yet routed, available only in schematic view.

Board Layer: Shows the Printed Circuit Board, available only in PCB view.

Silkscreen Bottom Layer: This layer is in development (N/A), available only in PCB view.

Silkscreen Bottom (Parts Label) Layer: Same as Silkscreen Bottom Layer.

Copper Bottom Layer: Shows the copper around the connections, available only in PCB view.

Copper Top Layer: This layer is in development (N/A), available only in PCB view.

Silkscreen Top Layer: Shows images (AKA silkscreen) on PCB which indicate where parts are placed, text and images, available only in PCB view.

Silkscreen Top (Parts Label) Layer: In order to show the parts labels of a particular part right click on the part and select “show part label”, shows all images (AKA silkscreen) on PCB of parts labels, available only in PCB view.

Part Image Layer: This layer is in development (N/A), available only in PCB view.

Rat's Nest Layer: Shows the most direct route between connected parts, available only in PCB view.

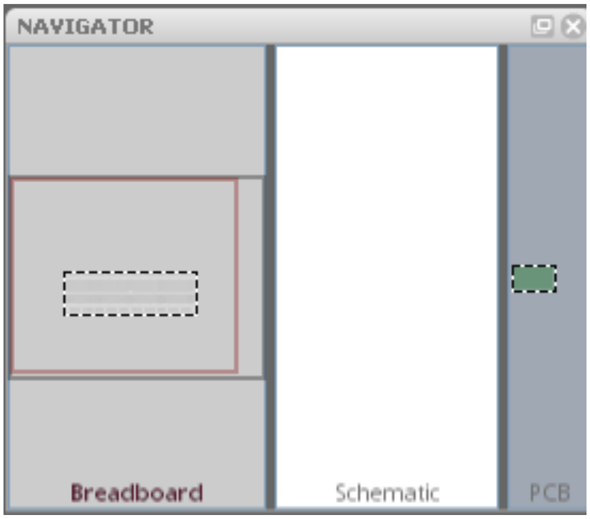
Notes Layer: Shows notes, available in all views.

Rulers Layer: Shows rulers, available in all views.

Fritzing Navigator Menu

The Navigator menu is where you can switch between the three different views; Breadboard, Schematic and PCB.

Another cool aspect of Fritzing is that when you add or rearrange wires (or traces) in one view, you can see the results in the navigator menu images of the other views. So if you are paying attention you can tell when you have altered your original circuit in a view other than the Breadboard view.



Breadboard view: Click on this section to display the breadboard view.

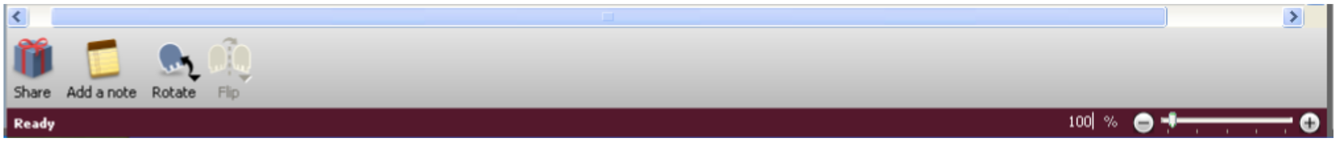
Schematic view: Click on this section to display the schematic view.

PCB view: Click on this section to display the PCB view.

Note: The Fritzing Parts, Inspector, Layer and Navigator windows are all re-sizable. It is also possible to undock these menu windows and place them where ever you like.

View Menu Options

The three different views (Breadboard, Schematic and PCB) have various menu options below them. These view menu options are explained below.



Share: Upload your project to the Fritzing website for help, comments, or just to share how awesome your project is! Available in all views.

Add a note: Add a note about your project. The note will only display in the view you add it to. Available in all views.

Rotate: Rotate a part, available in all views.

Flip: Flip a part, available in breadboard and schematic views. Not available for all parts.

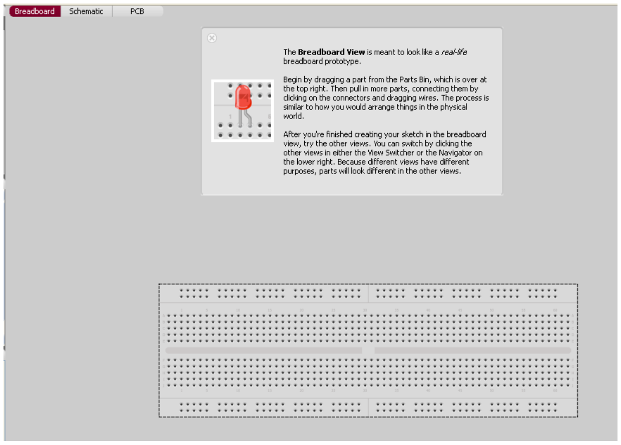
Autoroute: Place electrical traces automatically, the computer will only autoroute existing wire connections on your breadboard. Note that autoroute will leave connections unrouted and cross wires, so always double check this! You will need to route by hand. Not available in breadboard view.

Export Etchable PDF: Export a PDF with the traces and vias, for do it yourself PCB creation.

Percentage slider (set to 100% in the image above): Zoom, available in all views.

Fritzing Breadboard View

The first of the three views in Fritzing is the Breadboard View. This is where you will recreate the physical prototype you created using your breadboard, Arduino, wires and components. In order to use Fritzing effectively it is important to actually create your physical prototype on a real live breadboard before using Fritzing. Believe me, it will save you a world of hassle in the long run.

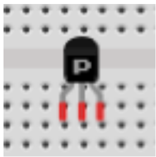
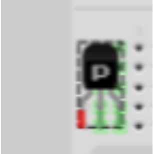
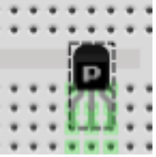


The **Breadboard View** is meant to look like a *real-life* breadboard prototype.

Begin by dragging a part from the Parts Bin, which is over at the top right. Then pull in more parts, connecting them by clicking on the connectors and dragging wires. The process is similar to how you would arrange things in the physical world.

After you're finished creating your sketch in the breadboard view, try the other views. You can switch by clicking the other views in either the View Switcher or the Navigator on the lower right. Because different views have different purposes, parts will look different in the other views.

Placing parts: To place parts click and drag from your parts bin to the breadboard, when the part is placed correctly and the leads of the piece are inserted in holes the leads will turn green instead of red. Pay close attention to this step, it is possible to correctly place some leads but not all the leads. Once a part is placed correctly all the other breadboard holes that are connected to the part will be highlighted in green. These green holes are where you can connect wires to provide an electrical path for this lead. Right click on placed parts for additional options.

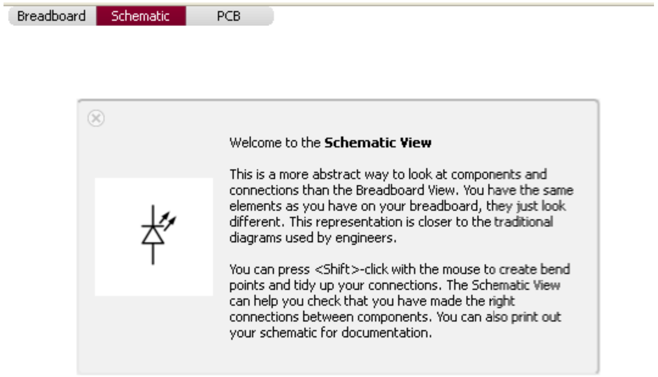
Incorrectly placed part	Sort of correctly placed part	Correctly placed part
		

Placing wires: To place wires click on a breadboard hole and drag. To move the wire click and drag either end. To create a bend in the wire double-click or click and drag anywhere along the wire other than the two ends. Right click on the wire for additional options.

Connections: To see all the wires and breadboard holes that are connected to any given point on the breadboard click and hold on that point. Any breadboard holes or wires that are connected will highlight in yellow.

Fritzing Schematic View

The second of the three views in Fritzing is the Schematic View. This is where you can see the schematic version of the circuit you are creating in the breadboard view. This view is nice if you are familiar with schematics because everything is simplified and you can check your connections easily.



Welcome to the **Schematic View**

This is a more abstract way to look at components and connections than the Breadboard View. You have the same elements as you have on your breadboard, they just look different. This representation is closer to the traditional diagrams used by engineers.

You can press <Shift>-click with the mouse to create bend points and tidy up your connections. The Schematic View can help you check that you have made the right connections between components. You can also print out your schematic for documentation.

Nets layer: This layer displays the connections you created using wires in the Breadboard view. These connections, known as nets, are unrouted, meaning they do not have a copper trace. The schematic view is where you will create the first set of routes. Nets are displayed as thin lines of different colors. When you first enter the schematic view after creating a circuit in the Breadboard view all connections are displayed as nets and need routing. To create a route, simply click and drag from the two ends of the components you want to connect. Once the connections have been routed the net will no longer be visible because a thicker wire line will overlay it.

Wires layer: This displays the routes that have been created. Connected route ends are highlighted in green, unconnected route ends are highlighted in red.

Autoroute: While Fritzing 0.4.3's autorouter is a step up from the original, you will need to double check the routes it creates as well as creating some on your own. To the right of the autoroute button there is a status field that tells you how many nets have been routed and how many are still unconnected.

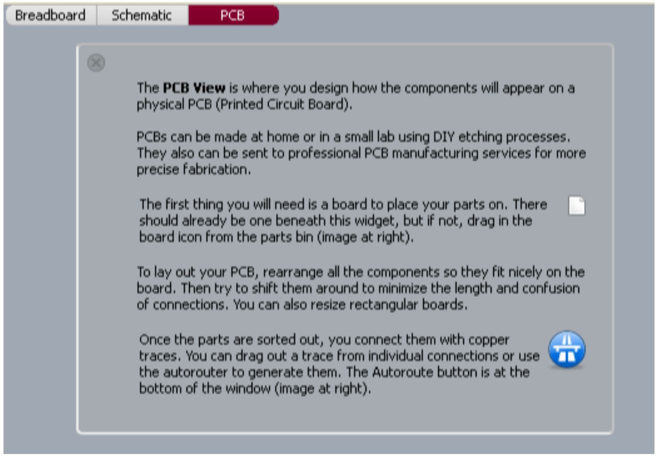
Hand routing: Let's face it, computers aren't all that great at creative solutions, you're going to have to create your own net routing time and time again. I promise it will get easier as you do it more often until it requires little to no thought. To make it easier for others to read your schematic make sure that the routes do not cross each other unintentionally.

Unconnected parts: It is possible to place parts in the Breadboard view without connecting them. Unconnected parts will appear highlighted in red. To connect them in the Schematic view simply click and drag. This is not recommended as it will change your breadboard circuit without your knowledge. It is extremely confusing when you switch back to the Breadboard view and discover your circuit wires look nothing like the wires you originally placed.

Additional options: Right click on nets, wires and parts for more options including changing the circuit, autorouting a single net at a time and deleting wires and nets. Be careful with this option because it will actually delete wires and traces from the other views.

Fritzing PCB View

The third of the three views in Fritzing is the PCB View. This is where you can see the Printed Circuit Board version of the circuit you are creating in the breadboard view. This is what the actual finished Printed Circuit Board will look like before you solder the components onto it.



The **PCB View** is where you design how the components will appear on a physical PCB (Printed Circuit Board).

PCBs can be made at home or in a small lab using DIY etching processes. They also can be sent to professional PCB manufacturing services for more precise fabrication.

The first thing you will need is a board to place your parts on. There should already be one beneath this widget, but if not, drag in the board icon from the parts bin (image at right).

To lay out your PCB, rearrange all the components so they fit nicely on the board. Then try to shift them around to minimize the length and confusion of connections. You can also resize rectangular boards.

Once the parts are sorted out, you connect them with copper traces. You can drag out a trace from individual connections or use the autorouter to generate them. The Autoroute button is at the bottom of the window (image at right).

The PCB: The green rectangle that represents the actual board you are placing part “footprints” (the electrical contacts you will solder the parts to) on. The PCB is also a part you can find in the core parts bin, it looks like a green square.

Arranging parts: PCB manufacturers usually charge a flat rate plus an amount that depends on the size of your PCB, so it's a good idea to try and create as small a PCB as possible. Even if you are planning on etching your own board using Ferric Chloride (talk about a cool science project!) smaller traces will use less of your materials and time, as well as creating less frustration. Rearranging parts in the PCB view will not effect the parts in the other views.

Traces: Traces are the copper paths that connect the various parts on your PCB. In the PCB view unconnected traces look like plain black lines running from part to part. Every single unconnected trace should be turned into a connected trace by clicking and dragging from one end of the unconnected trace (at the connector) to the other end of the trace (the other connector). Clicking once in the middle of the unconnected trace will also create a trace, but it will put a bend in the trace as well.

Connector: Connectors are the circles in parts that connect the part to the board and traces. Connectors with any type of trace (connected or unconnected) are displayed as green dots with yellow outlines. Connectors with no traces are displayed as red dots with yellow outlines, pay special attention to these connectors because usually all connectors need traces. To see what connectors are directly connected to each other click and hold on a connector. All other attached connectors will highlight in yellow.

Silkscreen layer: The silkscreen layer is where part outlines, text, and images are displayed. For example if you are creating a PCB to control an autonomous marshmallow catapult you might place the text “Catapult Brain v. 2.0” along with part outlines on your PCB so anyone putting together the board will know where to put the parts as well as what the board is used for.

Fritzing PCB View II

Special Parts and Tools to Note

Jumpers: Sometimes there is no way to connect all the traces on a board without crossing traces over each other, which will short out the traces. To avoid this use a jumper. A jumper is two connectors that are left untraced so later you can attach a wire “jumping” from one connector over the trace to the connector on the other side. To place a jumper click and drag the jumper to where you wish to cross a trace, then click and drag from one connector on the jumper to the connector on the part. Do this with both sides of the jumper. If you have correctly placed the jumper it should look like a blue line crossing a trace. When you click on one side of the jumper all traced connectors will highlight in yellow, indicating you have correctly placed the jumper.	 **Jumper**
Vias: Vias are holes with copper lining the inside. On two sided PCB designs vias are used in place of jumpers to switch a trace from one side of the board to the other. To place a via simply click and drag the part to where you want a via.	 **Via**
Silkscreen text: Often a PCB designer will need to put some text on the PCB for one reason or another. The silkscreen text part is designed for that purpose. To place silkscreen text click and drag the Silkscreen text part onto your board. Use the Inspector menu to change the text displayed. This part can also be used to place images.	 **Text**
Silkscreen image: If you've got a sweet logo you want to put on your PCB this is the part you will use to place it. To place a silkscreen image click and drag this part onto your board. Click on “Load Image” in the Inspector menu to load an image you have created for the board, then select the image from the drop down menu just above the “Load Image” button.	 **Image**

Design Rules Check: To use the Design Rules Check click on the Trace menu and select Design Rules Check. The Design Rules Check checks your PCB design for overlapping parts and traces. The tool displays the number of overlapping parts (includes parts and wires) in text below the PCB View. This tool will not check for overlapping traces that have not yet been connected. Make sure to run this tool before you consider your design finished.

Programming window: To open the Programming window click on the Window menu and select “open Programming window”. The Programming window is an experimental feature in development. It can be used for Picaxe and Arduino programming. This window plays the same role as the Arduino Environment, but is not intended to replace it.

Copper fill: To create a copper fill click on the Trace menu and select Copper Fill. To remove a copper fill click on the Trace menu and select Remove Copper Fill. A copper fill is a layer of copper inside the PCB. You can treat this layer like a huge wire, in more advanced PCB layout software it is often used as a ground connection. It is possible to connect existing traces to the copper fill by moving the trace after creating a copper fill. Be careful with this option though because if you refill the copper it will insulate all wires from this layer.

Fritzing PCB View III

Routing Tips and Tricks (From the Fritzing Website)

Place the parts with the most connections in the middle of the board.

Try to get short connections by moving and rotating parts.

Use the highlighting of equipotential connectors feature.

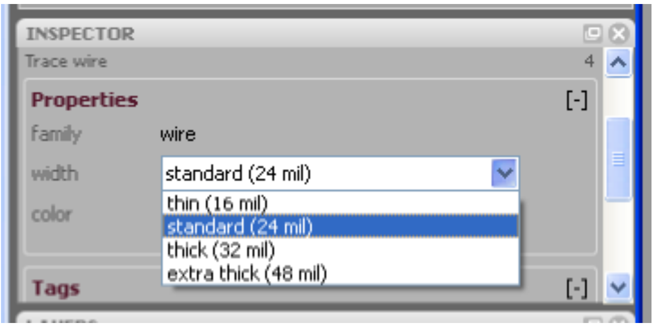
Add bend points for tidy routing and so that lines do not cross.

Don't forget the traces can go under parts like resistors.

Use jumper wires instead of watching the auto-route go crazy.

To achieve a better and nicer design, you would need to edit traces by moving, adjusting width and adding bend points. Width adjustment can be done in the Inspector. Please note that thin traces might ruin a DIY PCB production (if the traces are too thin the electrical current will not be able to flow properly), so keeping traces in medium thickness is safer.

Adjusting Trace and Wire Width

To adjust the trace or wire width click on the trace and select Width in the Properties submenu of the part Inspector Menu.	

family: wire

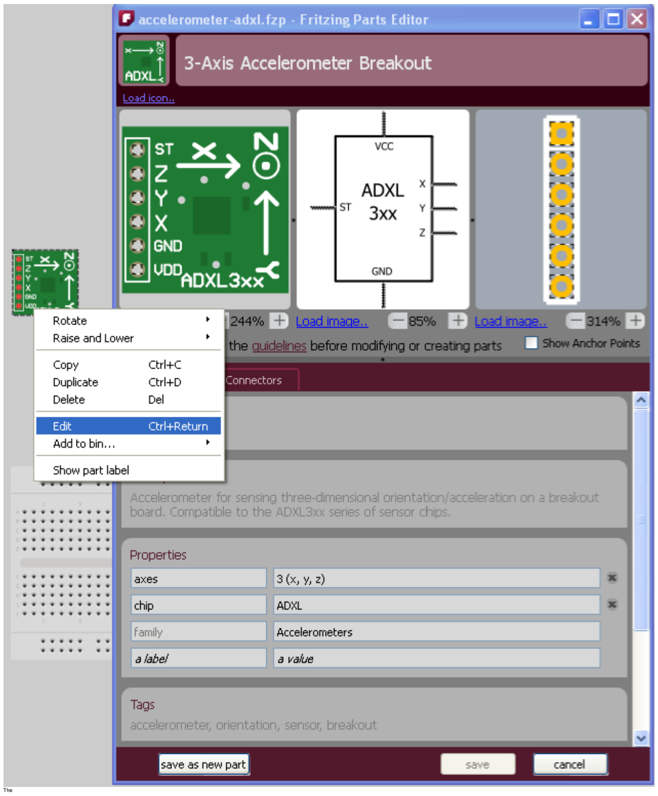
width: standard (24 mil)

Editing Existing Parts in Fritzing

Once you have become familiar with Fritzing and virtual prototyping you may find that a part you are working with on your physical breadboard is not present in the Fritzing library. Your first move should be to see if you can find a part in the existing library that is similar to the part you are working with. If you can find a similar part, then all you need to do is edit the existing virtual part so it matches the one you are working with.

To do this drag and drop the similar part to your view window. Open the part editor by either right clicking on the part and selecting Edit or selecting Edit under the Part menu.

The pop-up Fritzing Parts Editor window will look like the example to the right. If you have an internet connection definitely check out the guidelines for editing and creating parts.



the guidelines before modifying or creating parts

Accelerometer for sensing three-dimensional orientation/acceleration on a breakout board. Compatible to the ADXL3xx series of sensor chips.

Properties

axes	3 (x, y, z)
chip	ADXL
family	Accelerometers
a label	*a value*

Tags

accelerometer, orientation, sensor, breakout

Images: Load images you have created for each of the three views at the top of the Parts Editor. The image on the left should look like the physical part, the image in the center should be a schematic representation of the part (all leads must be shown) and the image on the right is simply an image of all the leads present on the part. The leads on these images are very important, without the proper connections your PCB will not work. You may also wish to include a silkscreen image of the part so you do not try to place the footprints too close together on the PCB. If the parts are too close together you will not be able to physically fit them next to each other on the PCB.

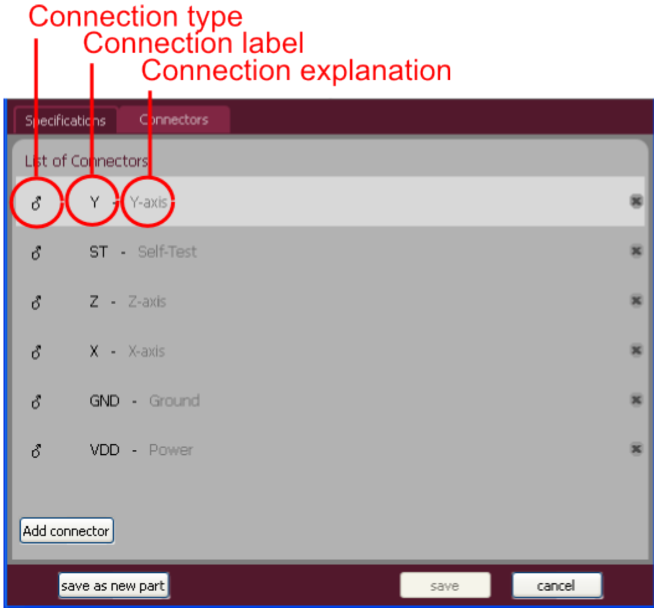
Specifications: Various text fields that provide information about the part. These do not actually effect the virtual part, but provide information about the part for users.

Connectors: This is the most important portion of editing a part. The next page is dedicated to the connectors menu.

Saving: Each sub-menu inside of the part editor will ask you to either save or cancel the changes you make. At the very bottom of the parts editor are the buttons labeled Save as New Part, Save and Cancel. When editing an existing part always Save as New Part in order to avoid permanently changing the part in case you will need it later in the prototyping process.

Editing Connector Parts in Fritzing

Connectors are very important for PCB layout. The connectors are where the parts are actually connected to the PCB and traces. Make sense, huh? Because connectors are so important you need to make sure that the information about the connectors in the parts you edit or create are correct and in the proper places.



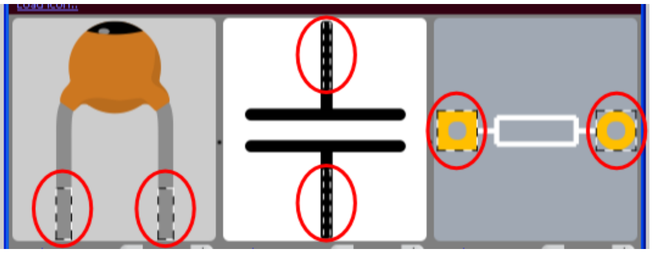
♂ Y - Y-axis
♂ ST - Self-Test
♂ Z - Z-axis
♂ X - X-axis
♂ GND - Ground
♂ VDD - Power

Connection type: The type of connection present on the physical part. The three options are Male, Female and Pad. Pad is the connection used in surface mount soldering.

Connection label: The label on the part label layer, often an abbreviation.

Connection explanation: An explanation of the connection label.

The images in the Fritzing Parts Editor indicate the location of the connectors with bounding boxes, circled in red in the image to the right. You can move and resize these bounding boxes to reflect the physical connectors.



Creating New Parts in Fritzing

The process of creating new parts in Fritzing uses the exact same set of menus and options that editing an existing part does, only you will need to fill in every single field and compare every connector to the physical part because the New Part fields start completely blank.

For more help with editing and creating parts:

Online Fritzing Libraries: http://fritzing.org/parts/
Help with creating parts: http://fritzing.org/learning/tutorials/creating-custom-parts/

From Fritzing to Physical PCB

OK, you've checked and double checked your virtual prototype to make sure it matches the physical prototype on your breadboard. You've checked and double checked the PCB layout in Fritzing and you're ready to take the final step of virtual prototyping; exporting PCB layout files so you can create a physical PCB!

There are two ways you can export the necessary files to create your physical PCB. The standard option is to export Gerber files, zip them and send them to a PCB prototype facilitator such as BatchPCB. A facilitator allows you, the inventor (that's right you are now officially an inventor), to purchase your prototype boards one at a time instead of five or ten at a time, because, let's face it, there might be some errors.

Exporting files for a PCB prototype facilitator: Create a folder to hold the Gerber files and only the Gerber files. Go to the File drop down menu, click on Export and select To Gerber. This will pop-up a window asking you where you would like to save the files, find the folder you created, select it and click OK. If you have not already created a folder there is also a Create Folder option. This will create five Gerber files and a text file. Zip all these files and send them to your friendly PCB prototype facilitator. Sit tight for a couple weeks and when your PCB prototypes show up in the mail, if you have the parts on hand, you're ready to start soldering everything together!

The second option for PCB prototype creation is to export an etchable PDF file and create the PCB yourself. No matter how you attempt this second option it requires some fairly advanced technology.

Exporting files for etching your own PCB: This first step is pretty easy, just click on the Export Etchable PDF button in the PCB view and save wherever you like. Actually etching the PCB is a whole different topic, here's a link to get you started:

For help etching your own PCBs:
Fritzing tutorial: http://fritzing.org/learning/tutorials/pcb-production-tutorials/diy-pcb-etching/

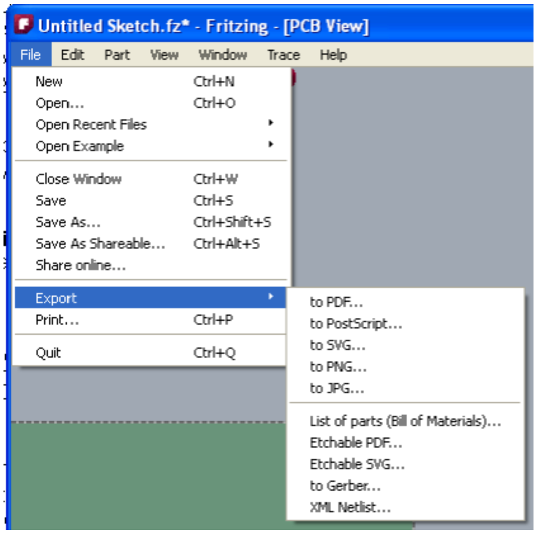
Other export file type options:

PDF, PostScript, SVG, PNG and JPG:
Image file formats.

List of parts:
Text file format listing the parts and components necessary to create your prototype.

Etchable SVG:
Similar to Etchable PDF, a different image file format.

XML Netlist:
Code file format.



Common Core Middle School Math Standards

Standard	Circuits	MS	HS	PWM	V drop & resistance
4.OA.1	10	2, 3	1, 8	yes	v drop
4.OA.2	6, 9			yes	
4.OA.3					
4.OA.4					
4.OA.5	10	2, 3	1, 8, 10	yes	v drop
4.NBT.1					
4.NBT.2					
4.NBT.3					
4.NBT.4					
4.NBT.5					
4.NBT.6					
4.NF.1				yes	
4.NF.2			8	yes	
4.NF.3.a		2		yes	v drop
4.NF.3.b		2		yes	
4.NF.3.c					
4.NF.3.d					
4.NF.4.a					
4.NF.4.b					
4.NF.4.c					
4.NF.5					
4.NF.6					
4.NF.7					
4.MD.1		2, 3, 7	1, 10, 13, 14	yes	
4.MD.2					
4.MD.3					
4.MD.4					
4.MD.5.a					
4.MD.5.b					
4.MD.6					
4.MD.7					
4.G.1					
4.G.2					
4.G.3					
5.OA.1	10				
5.OA.2				yes	
5.OA.3					
5.NBT.1					
5.NBT.2					
5.NBT.3.a					
5.NBT.3.b					
5.NBT.4					
5.NBT.5					
5.NBT.6					

Common Core Middle School Math Standards

Standard	Circuits	MS	HS	PWM	V drop & resistance
5NBT.7					
5.NF.1					
5.NF.2				yes	
5.NF.3					
5.NF.4.a					
5.NF.4.b					
5.NF.5.a		2, 3	8	yes	v drop
5.NF.5.b					
5.NF.6					
5.NF.7.a				yes	
5.NF.7.b					
5.NF.7.c			10	yes	
5.MD.1	10	1, 2, 3, 6, 7	1, 10, 11, 13, 14	yes	
5.MD.2				yes	
5.MD.3.a					
5.MD.3.b					
5.MD.4					
5.MD.5.a					
5.MD.5.b					
5.MD.5.c					
5.G.1					
5.G.2					
5.G.3					
5.G.4					
6.RP.1		2, 3	8	yes	
6.RP.2			8, 10	yes	
6.RP.3.a			8	yes	
6.RP.3.b					
6.RP.3.c		2, 3	8	yes	
6.RP.3.d		1, 2, 3, 4, 6, 7	1, 8, 10, 11, 13, 14	yes	
6.NS.1					
6.NS.2					
6.NS.3					
6.NS.4					
6.NS.5					
6.NS.6.a					
6.NS.6.b					
6.NS.6.c					
6.NS.7.a					
6.NS.7.b					
6.NS.7.c					
6.NS.7.d					
6.NS.8					

Common Core Middle School Math Standards

Standard	Circuits	MS	HS	PWM	V drop & resistance
6.EE.1		1, 2, 3, 4, 6, 7	1, 10, 11, 13, 14	yes	
6.EE.2.a		1, 2, 3, 4, 6, 7	1, 10, 11, 13, 14	yes	
6.EE.2.b					
6.EE.2.c		1, 2, 3, 4, 6, 7	1, 10, 11, 13, 14	yes	
6.EE.3					v drop
6.EE.4					
6.EE.5					
6.EE.6		1, 2, 3, 4, 6, 7	1, 10, 11, 13, 14	yes	v drop
6.EE.7		1, 2, 3, 4, 6, 7	1, 10, 11, 13, 14	yes	
6.EE.8					
6.EE.9		1, 2, 3, 4, 6, 7	1, 10, 11, 13, 14	yes	v drop
6.G.1					
6.G.2					
6.G.3					
6.G.4					
6.SP.1					
6.SP.2					
6.SP.3					
6.SP.4					
6.SP.5.a					
6.SP.5.b					
6.SP.5.c					
6.SP.5.d					
7.RP.1					
7.RP.2.a	10	1, 2, 3, 4, 6, 7	1, 2, 8, 10, 14	yes	
7.RP.2.b				yes	
7.RP.2.c	10	1, 2, 3, 4, 6, 7	1, 2, 8, 10, 14	yes	v drop
7.RP.2.d				yes	
7.RP.3	10	2, 3	10, 14	yes	
7.NS.1.a					
7.NS.1.b					
7.NS.1.c					
7.NS.1.d					
7.NS.2.a					
7.NS.2.b					
7.NS.2.c					
7.NS.2.d					
7.NS.3					
7.EE.1					

Common Core Middle School Math Standards

Standard	Circuits	MS	HS	PWM	V drop & resistance
7.EE.2		1, 2, 3, 4, 6, 7	1, 11, 13, 14	yes	
7.EE.3	10	1, 2, 3, 4, 6, 7	1, 11, 13, 14	yes	
7.EE.4.a					
7.EE.4.b					
7.G.1					
7.G.2					
7.G.3					
7.G.4					
7.G.5					
7.G.6					
7.SP.1					
7.SP.2	10				
7.SP.3					
7.SP.4					
7.SP.5					
7.SP.6					
7.SP.7.a					
7.SP.7.b					
7.SP.8.a					
7.SP.8.b					
7.SP.8.c					
8.NS.1					
8.NS.2					
8.EE.1					
8.EE.2					
8.EE.3					
8.EE.4					
8.EE.5				yes	
8.EE.6					
8.EE.7.a	10		10	yes	
8.EE.7.b					
8.EE.8.a					
8.EE.8.b					
8.EE.8.c					
8.F.1	10	2, 3	8, 10	yes	
8.F.2				yes	
8.F.3				yes	
8.F.4			8, 10	yes	
8.F.5				yes	
8.G.1.a					
8.G.1.b					
8.G.1.c					
8.G.2					
8.G.3					

Common Core High School Math Standards

Standard	Circuits	Middle S. Worksheets	High S. Worksheets	PWM	Add. Electrical
N.RN.1					
N.RN.2					
N.RN.3					
N.Q.1	10	1, 2, 3, 4, 6, 7, 12, 14	1, 3, 6, 8, 10, 11, 12, 13, 14	yes	resist., v drop
N.Q.2			8, 10, 13,	yes	
N.Q.3	10	1, 2, 3, 4, 6, 7, 12	1, 3, 8, 10, 11, 12, 13, 14,	yes	resist, v drop
N.CN.1					
N.CN.2					
N.CN.3					
N.CN.4					
N.CN.5					
N.CN.6					
N.CN.7					
N.CN.8					
N.CN.9					
N.VM.1					
N.VM.2					
N.VM.3					
N.VM.4.a					
N.VM.4.b					
N.VM.4.c					
N.VM.5.a					
N.VM.5.b					
N.VM.6					
N.VM.7					
N.VM.8					
N.VM.9					
N.VM.10					
N.VM.11					
N.VM.12					
A.SSE.1.a	10	1, 2, 3, 4, 6, 7	1, 3, 8, 10, 11, 13, 14	yes	v drop
A.SSE.1.b					
A.SSE.2		1, 2, 3, 4, 6, 7	1, 3, 8, 10, 11, 13, 14		v drop
A.SSE.3.a					
A.SSE.3.b					
A.SSE.3.c					
A.SSE.4					
A.APR.1					
A.APR.2					
A.APR.3					
A.APR.4					
A.APR.5					
A.APR.6					
A.APR.7					
A.CED.1	10	10	3, 8, 12	yes	
A.CED.2	10	1, 2, 3, 4, 6, 7	1, 8, 10, 13, 14	yes	resist
A.CED.3	9, 14		9		

Common Core High School Math Standards

Standard	Circuits	Middle S. Worksheets	High S. Worksheets	PWM	Add. Electrica
A.CED.4		1, 2, 3, 4, 6, 7	1, 10, 11, 13, 14	yes	
A.REI.1	10	1, 2, 3, 4, 6, 7	1, 8, 10, 11, 13, 14	yes	
A.REI.2					
A.REI.3			10	yes	
A.REI.4.a					
A.REI.4.b					
A.REI.5	10		10	yes	
A.REI.6					
A.REI.7					
A.REI.8					
A.REI.9					
A.REI.10					
A.REI.11					
A.REI.12					
F.IF.1	10		8, 9, 10	yes	
F.IF.2			8, 10	yes	
F.IF.3				yes	
F.IF.4				yes	
F.IF.5				yes	
F.IF.6					
F.IF.7.a				yes	
F.IF.7.b					
F.IF.7.c					
F.IF.7.d					
F.IF.7.e					
F.IF.8.a					
F.IF.8.b					
F.IF.9					
F.BF.1.a					
F.BF.1.b					
F.BF.1.c					
F.BF.2					
F.BF.3					
F.BF.4.a					
F.BF.4.b					
F.BF.4.c					
F.BF.4.d					
F.BF.5					
F.LE.1.a					
F.LE.1.b					
F.LE.1.c					
F.LE.2					
F.LE.3					
F.LE.4					
F.LE.5					
F.TF.1					
F.TF.2					
F.TF.3					
F.TF.4					

Common Core High School Math Standards

Standard	Circuits	Middle S. Worksheets	High S. Worksheets	PWM	Add. Electrica
F.TF.5					
F.TF.6					
F.TF.7					
F.TF.8					
F.TF.9					
G.CO.1					
G.CO.2					
G.CO.3					
G.CO.4					
G.CO.5					
G.CO.6					
G.CO.7					
G.CO.8					
G.CO.9					
G.CO.10					
G.CO.11					
G.CO.12					
G.CO.13					
G.SRT.1.a					
G.SRT.1.b					
G.SRT.2					
G.SRT.3					
G.SRT.4					
G.SRT.5					
G.SRT.6					
G.SRT.7					
G.SRT.8					
G.SRT.9					
G.SRT.10					
G.SRT.11					
G.C.1					
G.C.2					
G.C.3					
G.C.4					
G.C.5					
G.GPE.1					
G.GPE.2					
G.GPE.3					
G.GPE.4					
G.GPE.5					
G.GPE.6					
G.GPE.7					
G.GMD.1					
G.GMD.2					
G.GMD.3					
G.GMD.4					
G.MG.1					
G.MG.2					
G.MG.3					

Common Core High School Math Standards

Standard	Circuits	Middle S. Worksheets	High S. Worksheets	PWM	Add. Electrica
S.ID.1				yes	
S.ID.2					
S.ID.3					
S.ID.4					
S.ID.5					
S.ID.6.a					
S.ID.6.b					
S.ID.6.c					
S.ID.7					
S.ID.8					
S.ID.9					
S.IC.1					
S.IC.2					
S.IC.3					
S.IC.4					
S.IC.5					
S.IC.6					
S.CP.1					
S.CP.2					
S.CP.3					
S.CP.4					
S.CP.5					
S.CP.6					
S.CP.7					
S.CP.8					
S.CP.9					
S.MD.1					
S.MD.2					
S.MD.3					
S.MD.4					
S.MD.5.a					
S.MD.5.b					
S.MD.6					
S.MD.7					

Advanced Section

This section will discuss a few things that we didn't cover in the rest of the binder; more advanced programming concepts, additional data types and data structures, powering your projects, and interfacing with the Processing environment.

Arrays: If variables can be thought of as buckets that hold a single piece of information, then arrays can be thought of as a collection of buckets, or a big bucket with a lot of little buckets inside. Arrays are extremely useful for a lot of different programs – basically, any time you want to perform a similar operation on several variables (of the same type) – you should consider putting the variables in an array. For example, if I want to blink eight LED's at the same time, I could put them in an array, and then use a for loop to iterate over the array, like so:

```
/* this is the array that holds the pin numbers our LED's would be
connected to */

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


// in setup()we can set all the pins to output with a simple for loop

// 8, because we have 8 elements in the array

for( int i = 0; i < 8; i++) {

// sets each ledPin in our array to OUTPUT

 pinMode(ledPins[i], OUTPUT);

}
```

The `ledPins[i]` part is important; it allows us to reference each element in our array by its place in the array, starting with 0 (which can be confusing). So, in our example above ledPins[0] == 2, since 2 is the 1st element we put into the array. This means that ledPins[1] == 3, and ledPins[7] == 9, and is the last element in our array. If this doesn't make sense, don't worry.

See http://arduino.cc/en/Reference/Array for further explanation.

Float: Data type for floating point numbers (those with a decimal point). They can range from 3.4028235E+38 down to -3.4028235E+38. Stored as 32 bits (4 bytes). A word of advice: floating point arithmetic is notoriously unpredictable (e.g. 5.0 / 2.0 may not always come out to 2.5), and much slower than integer operations, so use with caution. Some readers may be familiar with the '**Double**' data type – currently, the Arduino implementation of Double is exactly the same as Float, so if you're importing code that uses doubles make sure the implied functionality is compatible with floats.

Long: Data type for larger numbers, from -2,147,483,648 to 2,147,483,647, and store 32 bits (4 bytes) of information.

String: On the Arduino, there are really two kinds of strings: strings (with a lower case 's') can be created as an array of characters (of type `char`). String (with a capital 'S'), is a String type object. The difference is illustrated in code:

```
Char stringArray[10] = "SparkFun";

String stringObject = String("SparkFun");
```

The advantage of the second method (using the String object) is that it allows you to use a number of built-in methods, such as length(), replace(), and equals().

More methods can be found here: http://arduino.cc/en/Reference/StringObject

Increment (++): Increment is an easy way to tell a number variable to add 1 to itself. So instead of writing `variable = variable + 1;` all you have to write is `variable++;`. It is commonly used in `for` loops like so:

```
for(int i = 0; i < 10; i++) {
//will increment i by 1 each time through the loop
}
```

Decrement (--): Basically the inverse of increment. Writing `variable--;` is the same as writing `variable = variable - 1;`

Compound Notation: Compound Notation is similar to increment and decrement in the sense that it provides a shorter way of performing arithmetic on a variable. Compound Notation can be used with addition(+=), subtraction(-=), multiplication(*=), and division(/=). For example:

```
float f = 10;
f += 10; // f now equals 20
f -= 5   // f now equals 15
f *=2;   // f now equals 30
f /=3;   // f now equals 10
```

Other Useful Arduino Functions:

delay(): The delay() function is very useful in programs where you want (you guessed it) a delay between actions (such as an LED blinking on and off). delay() takes its argument in milliseconds, so `delay(1000);` would be a 1 second delay.

millis(): The millis() function returns the number of milliseconds since the program started running (up to about 50 days, at which point it resets to 0).

This can be useful if you sketch requires a timer or reset. For example, if I needed to blink an LED every five minutes, I could write:

```
if (millis() % 300000 == 0) {
 digitialWrite(ledPin, HIGH);
 delay(1000);
 digitalWrite(ledPin, LOW);
}
```

random(): The random() function will give you a pseudo-randomly generated number, with either a maximum number, or a maximum and minimum. So, if I wanted random numbers between 1 and 100, I would write:

```
int randomNumber = random(1,100);
```

Useful Libraries

Libraries are a great and easy way to extend the functionality of your Arduino. They're basically collections of code that you can import into your sketch by going to 'Sketch'-> 'Import Library'. The Arduino software ships with several useful Libraries, some of which are discussed below. The full list of available libraries and documentation can be found at: http://arduino.cc/en/Reference/Libraries.

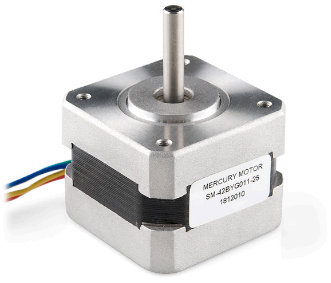
EEPROM: EEPROM stands for "Electrically Erasable Programmable Read-Only Memory", but you can think of it as the Arduino's permanent storage unit. Basically, if you want to keep track of something (say, sensor values) you can read or write them to the EEPROM, using the EEPROM library and the read() and write() functions. A complete example of reading in an analog sensor value and writing it to EEPROM can be found online at: http://arduino.cc/en/Tutorial/EEPROMWrite.

Ethernet: The Ethernet library, along with an Arduino Ethernet Shield, can enable your project to connect to the internet, and allows your project to act as either a server (accepting incoming connections) or a client (making outgoing connections). Check out the Arduino tutorial homepage under 'Ethernet Library' (http://www.arduino.cc/en/Tutorial/HomePage) for some code examples. You can find the Ethernet shield here: http://www.sparkfun.com/products/9026.

LiquidCrystal: This library is for controlling Liquid Crystal Displays (LCD's). It is designed for most text-based LCD's which use the Hitachi HD44780 chipset and driver – if that's greek to you, don't worry, there are many compatible screens out there. The basic wiring and code to get started are both pretty easy, and can be found here: http://arduino.cc/en/Tutorial/LiquidCrystal.

Stepper: Stepper motors are a type of motor that rotates continuously in small, precise steps. For this reason it makes a very useful component in any project that requires precise or continuous motion. The stepper library, in combination with a stepper motor board to control the motor (most steppers use more energy than the Arduino can provide) makes using stepper motors super easy.

Stepper Motors typically look something like this:



Stepper motors can be used pretty easily with a stepper motor driver like the EasyDriver, which you can find here (along with schematics and example code): http://www.sparkfun.com/products/10267.

Always remember to check the datasheet for any component you plan on using so you don't burn out your board or the motor.

Processing

Processing is a free, open-source programming language and environment created by Casey Reas and Ben Fry at the MIT Media Lab. Processing and Arduino are, in fact, built off some of the same code, and look very similar in their programming environments. However, while Arduino is built for interfacing with the physical Arduino boards, Processing specializes in visual-based programming on the computer. What's exciting about this is that Processing can read in values from Arduino on the Serial port and then manipulate visuals based on what the Arduino is doing. For example, you can make the turning of a potentiometer change the color of the screen in Processing. In order to do this, you need both Processing and Arduino set up properly.

You can download Processing from: http://processing.org/download/ (there are pretty good install instructions on the site, as well as a bunch of example code and references).

Now that you're all set with Processing and Arduino, let's set up a simple example where we read in the analog values from a potentiometer and change the background color in Processing in response.

First, set up Circuit-08 on your breadboard (we won't be using the LED though, so you can leave that out if you like). Next, open Arduino and type this code:

```
/*
AnalogReadSerial
Reads an analog input on pin 0, prints the result to the serial
monitor
This example code is in the public domain.
*/

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

void loop() {
 int sensorValue = analogRead(A0);
 Serial.print(sensorValue/4, BYTE);
}
```

Upload it to your Arduino board. We send the sensor value in byte form divided by four because we want Processing to be able to read in the values as integers from 0-255 (the full greyscale color range).

Now, open up Processing, and type in the following:

```
/* Analog Read from Arduino
 Change Background color based on analog sensor value
 This code is in the Public Domain
*/

import processing.serial.*; //import serial library

Serial myPort; //get a Serial object

void setup() {
 println(Serial.list()); //print out the available ports
 //pick the first port in our list
 myPort = new Serial(this, Serial.list()[0], 9600);
 size(600,600); //set the canvas size
}

void draw() {
}

void serialEvent(Serial myPort) {
//when the serial port gets a byte do the code below

 int in = myPort.read(); //read in a byte from the port
 background(in); //set the background color to the value
 println(in); //print it out to Processing's monitor
}
```

Launch the Processing sketch. Voila! Turning the pot should fade the color of the screen from white to black. If you don't understand all of the code, don't worry. Processing was built with new users in mind. There's great documentation on the Processing site: http://processing.org as well as a bunch of resources listed in the references section. Remember, this is just the tip of the iceberg; the only limit of what you can do is your imagination!

Further References:

Arduino:

Main site: http://arduino.cc

Reference (for functions): http://arduino.cc/en/Reference/HomePage

Arduino's Getting Started Guide: http://arduino.cc/en/Guide/HomePage

Examples: http://arduino.cc/en/Tutorial/HomePage

Tutorials: http://www.arduino.cc/playground/Learning/Tutorials

Forum: http://arduino.cc/forum/

DIY Electronics Tutorials:

Sparkfun: http://www.sparkfun.com/tutorials

Adafruit: http://www.adafruit.com/index.php?main_page=tutorials

LadyAda: http://www.ladyada.net/learn/arduino/

Instructables: http://www.instructables.com/tag/type-id/category-technology/channel-arduino/

Tom Igoe's Physical Computing Pages: http://tigoe.net/pcomp/index.shtml

Fritzing & DIY Circuit Board Production:

Main Fritzing site: http://fritzing.org/

Learning Fritzing: http://fritzing.org/learning/

Fritzing Part Libraries: http://fritzing.org/parts/

PCB Production: http://fritzing.org/learning/tutorials/pcb-production-tutorials/

BatchPCB: http://batchpcb.com/index.php/Products

Search on Instructables.com for 'Circuit Board' or 'PCB' – there are dozens of tutorials, one of which will probably suit your particular situation.

Books:

SparkFun's List: http://www.sparkfun.com/categories/176

Electrical Engineering 101: http://www.sparkfun.com/products/9458

Getting Started With Arduino: http://www.sparkfun.com/products/9301

Make: Electronics: http://www.sparkfun.com/products/9600

Making Things Move: http://www.sparkfun.com/products/10394

Making Thing Talk: http://www.sparkfun.com/products/9300

Processing:

Processing Website: http://processing.org

Open Processing: http://openprocessing.org

Learning Processing: http://learningprocessing.com

Creative Applications: http://www.creativeapplications.net/category/processing/

Other Stuff:

Illustrated Guide to Soldering (“Soldering is Easy: Here’s How to do it”): http://mightyohm.com/files/soldercomic/FullSolderComic_20110409.pdf

Evil Mad Science (blog and store): http://www.evilmadscientist.com/

Make: Magazine online (also has projects and videos): http://makezine.com/

Making Things Move Companion Website: http://www.makingthingsmove.com/

Arduino Cheat Sheet

(needs re-branding)

Glossary:

ADC: Analog to Digital Converter. Any method of converting an analog signal (a voltage) to a digital signal (a number). Here is the equation necessary to do this:

$$ADC\ Value = \frac{(Voltage\ on\ Pin(mV)) * (Max.\ ADC\ Value)}{(System\ Voltage(mV))}$$

Analog: A measurement or signal that has values between On and Off. Examples include voltage, pressure, any type of wave, and volume. One useful metaphor for teaching is the zipper (if you have a zipper on your jacket or hoodie). In comparison to the button (Digital) the zipper has many states between completely open and closed. Analog's counter part is Digital.

array: Arrays are a way to store several variables in a list, or array. In Arduino arrays are declared in a couple different ways. Example: *int arrayName [6];* this will create an array called arrayName with six spaces for variables inside it. A value of 100 would be assigned to the first space in arrayName like this: *arrayName [0] = 100;* (The first value is assigned to the 0 slot.) Here is an example of declaring and assigning values in an array at the same time: *arrayName [6] = {100, 150, 300, 50, 100, 120};* Here is an example of referencing the sixth value in the array arrayName: *arrayName [5]*.

Bias: A state describing voltage and current in a component.

Boolean: A variable type or form of logic based on the assumption that any given variable or state can only have one of two values; true or false, HIGH or LOW, 1 or 0.

bounce: Bounce occurs when a switch (or other type of input) attempts to change it's position to open or closed but does not stay in that position. Due to this you will see the electrical signal rising and falling when it should be at a constant value. If you’re experiencing bounce issues, try using the `delay( )` function.

button: A digital input with only two states; pressed or not pressed. Depending on the layout of the circuit these two states can correspond to either HIGH or LOW.

char: Variable type character, 8-bit size, any single symbol. Examples: A, a, 1, or !

comments: Used to write notes in code that are not part of the execution. For a single line use //, for a block of lines start with /* and end with */

constrain: A function used to constrain a value between a given range. Example: *sensVal = constrain (sensVal, 10, 150);* This constrains the sensVal value between 10 and 150. If it is under 10 sensVal is assigned 10, if it is over 150 sensVal is assigned 150.

current: This can refer to either the the flow of electrical charge or the rate of flow of electrical charge, measured in mA.

Digital: A measurement or signal that has only two values, On and Off. On and Off can also be expressed as HIGH and LOW, as well as 1 and 0. Examples include Boolean logic, open or closed and button state. One useful metaphor for teaching is the button (if you have a button on your jacket or hoodie). In comparison to the zipper (Analog) the button has only two states; connected and unconnected. Digital's counter part is Analog.

diode: A two terminal electrical component that only conducts in one direction.

Ground: In electrical engineering, ground or earth may be the reference point in an electrical circuit from which other voltages are measured, or a common return path for electric current, or a direct physical connection to the Earth.

float (signal): A signal due to a pin that is not attached to anything. A floating pin can read anything between HIGH and LOW.

float (data type): Variable type float, used for floating point operations (i.e. numbers with decimal points).

for: for(variable declaration and assignment; condition; variable increment){ }
A form of iteration, code inside the curly brackets will repeat until the condition is false. Example of a for loop that will loop four times:

```
for (i = 0, i < 4, i++) {//do this code each time};
```

forward bias: The state of a component describing voltage saturation necessary to activate a component.

footprint: A footprint is the pattern on a circuit board to which your parts are attached. This includes the electrical connections and silkscreen.

flyback diode: Used to reduce voltage spikes seen across inductive loads due to a sudden loss in voltage from the power source.

if statements: if(condition){ }
else if (condition) { }
else { }
This will execute the code inside the curly brackets if the condition is true, and if the condition is false it will test the "else if" condition. If the "else if" condition is true it will execute the code in the second set of curly brackets. Otherwise it will execute the code inside the third set of curly brackets.

Input: A signal or data received by a processor.

int: Variable type integer, 16-bit size, any number between -32,768 and 32,767.

iteration: When something happens over and over and over, but changes a little each time.

lead: Electrical contacts for parts, these usually look like wires or pins extending off the part.

LED: A light emitting diode.

Library: A collection of code that has been packaged so it can be included and then used in code. Servo example: `#include <Servo.h>` (This includes the library so it can be used) `Servo myservo;` (This creates a servo object so the functions inside the library can be used) `myservo.write(90);` (This uses the write function in the servo library, setting the servo's angle to 90 degrees.

loop ( ): Looks like- `void loop ( ){ }` The main loop in an Arduino sketch, this where the action happens. The `loop ( )` function is present in every single Arduino sketch. The Arduino will execute the code inside the curly brackets, once it has finished this code it will start over from the beginning of the loop function.

map: A function used to re-map a value between a range to a value between another given range. Example: `sensVal = map (sensVal, 10, 150, 100, 1500);` This maps the sensVal value from somewhere between 10 and 150 to somewhere between 100 and 1500 proportionally.

microcontroller: A tiny computer on a single integrated circuit with a core processor, memory and programmable input/output.

motor: An electrical component that converts electrical energy to mechanical energy.

Ohm's Law: V = I * R where V is Voltage, I is Current and R is Resistance. A handy way to figure out any one of these values given the other two. The complicated version: Ohm's law states that the current through a conductor between two points is directly proportional to the potential difference or voltage across the two points, and inversely proportional to the resistance between them.

operators: Similar to mathematical symbols, pay special attention to the difference between = and ==.

output: A signal or data transmitted by a processor.

piezo element: A digital component which moves a disc to one of two possible positions to create an analog output via vibration, often used to create annoying melodies with little aesthetic value.

Pin: A place to connect electrical circuits to one another.

pinMode: Arduino command used to set pins to either INPUT or OUTPUT.
Looks like- `pinMode ( pinNumber, value );` where pinNumber is the pin to be set and value is either INPUT or OUTPUT. pinMode can also be used to turn Analog In pins into Digital pins.

potentiometer: A voltage divider with a dial to change the values of the two resistors inside.

pseudo-code: A human-readable way of describing a computer program so that it is easier to understand. See the description of ‘if statements’ in the glossary for an example.

Pulse Width Modulation: A commonly used technique for controlling digital systems to create a simulated analog output. Often abbreviated as PWM.9

relay: An electrically operated switch.

Resistance: A measure of the opposition to electrical current. Measured in Ω (ohms).

resistor: Component used to restrict the amount of current that can flow through a circuit. Rated in Ω (ohms).

Serial: universal asynchronous receiver/transmitter.

Serial Monitor: Window in Arduino programming environment that allows the user to monitor serial communication.

setup (): Looks like- `void setup ( ){ }` All the code between the curly brackets will run once when the Arduino program first starts. (As well as each time it is restarted.)

shift register: In this case a chip which allows you to change the value of it's output pins, starting with either the first or last pin, by “shifting” a new value in and “shifting” all the old values towards the opposite end of the chip. The shift register uses three pins (latch, clock, and data) to control eight output pins.

sketch: The code created in your Arduino environment which is then saved onto your Arduino board to make something happen.

sketchbook: Folder where your sketches are stored.

temperature sensor: A sensor used to convert temperature to an analog reading, in this case a voltage, which can be read by your Arduino.

trace: A copper path on a PCB necessary for electrical conductivity between parts.

transistor: A semiconductor device used to amplify or control an electronic signal.

value: Worth or symbol stored in a variable.

variable: A symbolic name associated with a value and whose associated value may be changed.

voltage: The difference in electrical potential between any two given points of a circuit.

voltage saturation: When a component has the necessary voltage present to allow it to operate.