1. Introduction to functions

Today, we'll start with a simple programming method that will help keep your programs small, tidy, and easy to understand. Re-build the 4 LED circuit from the previous workshop, and open up the code for the blinking pattern. It might look something like this:

```
int ledPins[] = {2,3,4,5};       //An array to hold the pin each LED is connected to
                                 // For example, LED #0 is connected to pin 2
void setup() {
        for(int i = 0; i < 4; i++){              //this is a loop that will repeat 4 times
          pinMode(ledPins[i],OUTPUT);      //it sets the LED pins to output
        }
}

void loop() {

CODE FOR BLINKING PATTERN 1 HERE;

CODE FOR BLINKING PATTERN 2 HERE;

}
```

You can neaten up this code by creating your own functions. For example, if you have two different blinking patterns, you can put them into separate functions like this:

```
void blinky1() {
CODE FOR BLINKING PATTERN 1 HERE;
}

void blinky2() {
CODE FOR BLINKING PATTERN 2 HERE;
}
```

and then your loop function will just contain calls to the functions:

```
void loop(){

blinky1();      // calls the first blink pattern function

blinky2();      // calls the second blink pattern function

}
```

This is an extremely useful way of organizing your code into chunks, each of which perform a different task. Keep this idea in mind, and we'll come back to it.

2. 'I ain't afraid of no ghosts': Build your own EMF detector

a. Take a wire and plug it in to one of the Analog inputs. You've just built an antenna, the simplest possible sensor.

There are plenty of stray electric fields in your environment from cell phones, Wifi, radio waves, the electric wiring in the room, laptops, the lights, your fingers, and so on. Those of you who've taken some physics know that the electric fields apply forces on charged particles. So, picture the electrons in your wire. They're being wiggled up and down by these stray electric field in the room. And moving charges make up a *current*, which we can measure on the analog port. Technically, what you are doing here is measuring the induced voltage (or EMF - electromotive force, a terrible name) on the wire.

Here is some example code that will return the values measured by the analog port. It uses 'serial communication', which just means the Arduino is sending data to your computer, 1 bit at a time.

```
void setup(){
  Serial.begin(9600); // Tells Arduino to send back 9600 bits of data per second.
                      // 9600 is an old standard from the modem days
}

void loop(){
  int brightness = analogRead(0); //Read in the value from Analog Port 0
  Serial.println(brightness);     //And print it to the serial port
  delay(100);                     // Wait a tenth of a second.

}
```

b. Run the code, and open the Serial Monitor to see what's going on. You should see a string of numbers between 0 to 1023, that corresponds to the voltage being measured by the analog port. Move the antenna around the room, and note down the maximum and minimum values that you see.

c. Next, use the value of this EMF detector to control the blinking rate of your 4 LEDs. To do this, it will help to define a variable:

```
int delaytime = 100    // This variable stores the default delay which is 100 ms
```

Whenever the Arduino measures a reading from your antenna, you can use this value to set the blinking rate. The map function will come in handy here:

```
delaytime = map(sensorvalue, low, high, smalldelay, bigdelay);
```

d. The serial monitor is a rather inconvenient way of reading inputs. It would be nicer if we could plot a graph instead.

To do this, install *Processing* on your computer (http://processing.org/). Then, in the Processing environment, run the graphing code below. It's also available at http://arduino.cc/en/Tutorial/Graph
If your Arduino is plugged in and sending data to the computer, this code should create a plot of whatever the Arduino sees.

```
// Graphing sketch in PROCESSING
// This program takes strings from the serial port at 9600 baud and graphs them.
// It expects values in the range 0 to 1023, followed by a newline
// Created 20 Apr 2005 , Updated 18 Jan 2008 , by Tom Igoe
// This example code is in the public domain.

import processing.serial.*;

Serial myPort;       // The serial port
int xPos = 1;        // horizontal position of the graph

void setup () {
// set the window size:
size(400, 300);

// List all the available serial ports
println(Serial.list());
                                                    // I know that the first port in the serial list on my mac
                                                    // is always my  Arduino, so I open Serial.list()[0].
                                                    // Open whatever port is the one you're using.
myPort = new Serial(this, Serial.list()[0], 9600);
// don't generate a serialEvent() unless you get a newline character:
myPort.bufferUntil('\n');
// set inital background:
background(0);
}

void draw () {
// everything happens in the serialEvent()
}

void serialEvent (Serial myPort) {
// get the ASCII string:
String inString = myPort.readStringUntil('\n');

if (inString != null) {
        // trim off any whitespace:
        inString = trim(inString);
        // convert to an int and map to the screen height:
        float inByte = float(inString);
        inByte = map(inByte, 0, 1023, 0, height);

        // draw the line:
        stroke(127,34,255);
        line(xPos, height, xPos, height - inByte);

        // at the edge of the screen, go back to the beginning:
        if (xPos >= width) {
                xPos = 0;
```

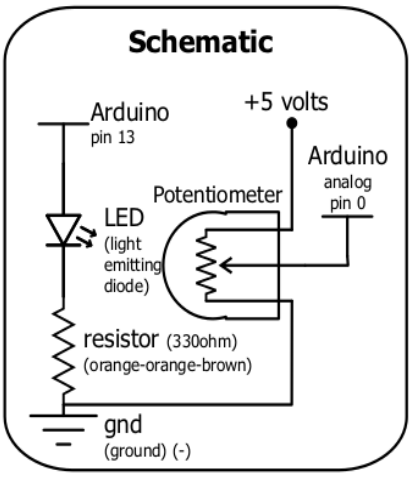
```
                background(0);
                 }
        else {
                 // increment the horizontal position:
                xPos++;
                }
        }
}
```

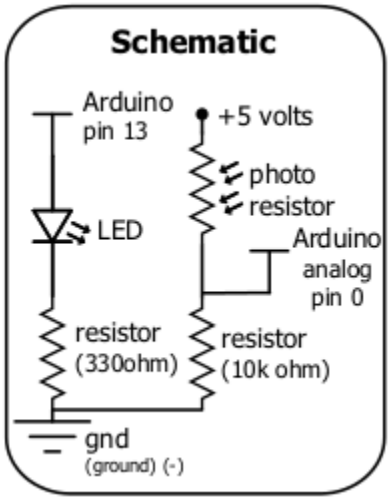
Most likely, the graph will vary over a smaller range than the entire possible range of 0 to 1023. There are two different ways you can fix this:
1. You can use the map function to rescale the range of your analog input variable.
sensorvalue = map(sensorvalue, low, high, 0, 1023);
2. You can change the range over which the processing code plots, by changing the numbers 0 and 1023 in the following line:
inByte = map(inByte, 0, 1023, 0, height);

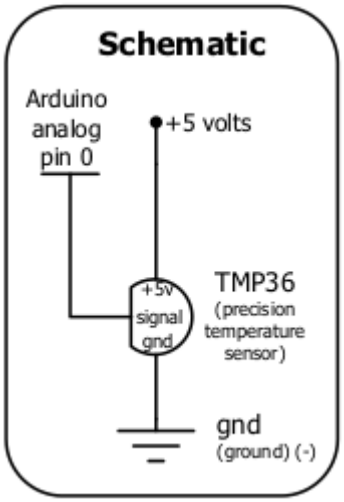
3. Sensing the World

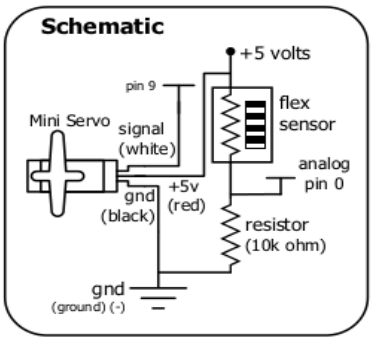
Wire up a sensor in a circuit, and plot the output using the Processing program from above. Do this for two different sensors. When you're done, hand the sensor back to me so another group can use it.

We've got a touch sensor, flex sensor, photoresistors (light sensor), temperature sensor, a gyroscope/accelerometer, and a potentiometer (dial) for you to try out. You may need the wiring diagrams for different sensors:









4. Use your mouse to dim the lights

We're going to learn how to send data from the computer to the Arduino. This is Serial communication going in the opposite direction. The new command here is **Serial.read()**

a. Hook up an LED to one of the pins capable of analog output. Try out the following code, which will let you control the LED brightness by typing in a key. Think through what this code is doing.
(You can get the code from http://arduino.cc/en/Tutorial/Dimmer)

```
int ledPin = 9;      // the pin that the LED is attached to
int brightness;

void setup()
{
  // initialize the serial communication:
  Serial.begin(9600);
  // initialize the ledPin as an output:
  pinMode(ledPin, OUTPUT);
}

void loop() {

  // check if data has been sent from the computer:
  if (Serial.available()) {
    // read the most recent byte (which will be from 0 to 255):
    brightness = Serial.read();
    // set the brightness of the LED:
    analogWrite(ledPin, brightness);

   Serial.println(brightness);
  }
}
```

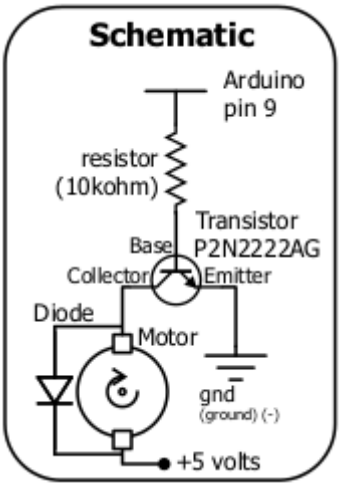
Start up the serial monitor and type in a letter. The Serial Monitor then converts this to a 1 byte number, based on its ASCII code. The Arduino should read this number, and set the brightness to the corresponding value.

b. Now try out the following Processing program, that takes the position of your mouse on a slider, and converts it into a byte that is sent to the Arduino. (You can get the code from http://arduino.cc/en/Tutorial/Dimmer)

```
// Dimmer - sends bytes over a serial port
// by David A. Mellis
//This example code is in the public domain.

import processing.serial.*;
Serial port;

void setup() {
size(256, 150);

println("Available serial ports:");
println(Serial.list());

// Uses the first port in this list (number 0).  Change this to
// select the port corresponding to your Arduino board.  The last
// parameter (e.g. 9600) is the speed of the communication.  It
// has to correspond to the value passed to Serial.begin() in your
// Arduino sketch.
port = new Serial(this, Serial.list()[0], 9600);

// If you know the name of the port used by the Arduino board, you
// can specify it directly like this.
//port = new Serial(this, "COM1", 9600);
}

void draw() {
// draw a gradient from black to white
for (int i = 0; i < 256; i++) {
stroke(i);
line(i, 0, i, 150);
}

// write the current X-position of the mouse to the serial port as
// a single byte
port.write(mouseX);
}
```

5. Rev up a motor

The Arduino's pins are great for controlling tiny electric devices like LEDs. But what if you need to power something more hefty, such as a toy motor, or a washing machine? An Arduino can't handle such large currents. This is where transistors come in handy. A transistor is basically a digital switch. It can switch on a lot of current using a much tinier current.

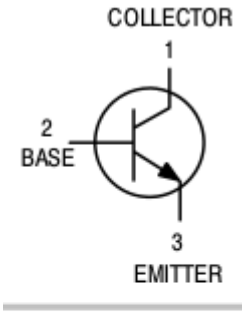


A transistor has 3 terminals. By sending a small current through the **base**, you 'unlock the gate', and a large current can flow from the **collector** to the **emitter**.

Wire up the motor as shown in the schematic. You will also put in a diode, which is essentially a one-way valve for current. This will prevent the motor getting damaged from stray currents that it can generate. Every diode has a thin black line, indicating that current can't flow in from that side.

What is the 10,000 Ohm resistor doing in the circuit? What is the maximum current that can flow in to the base terminal? Does it exceed the maximum rating of your transistor?

Check the packaging for the terminal labels for your transistor. If it came without packaging, see the figure on the right.



Once you have everything wired up, you can just treat your motor as if it were an LED. Stick a piece of tape to the motor shaft, so you can see how fast it is spinning.

Now for the code. Write a program to control the motor. Use functions to keep things simple and modular. Your loop function should look like this:


```
void loop() {
motoronandoff();
motoraccelerate();
motordeccelerate();
}
```

Your task is to fill in the code for these functions.

```
void motoronandoff(){
CODE GOES HERE;
}
```

and so on..

The accelerate function should be somewhat similar to your fade function. It takes the motor from zero speed to full speed. The deccelerate function does the opposite. Remember, for loops are your friends.