Data Logger (and using cool sensors!)

Part A. Writing to the Serial Monitor

a. Based on the readings from the serial monitor, what is the range of the analog values being read? 0 to 1023. **b. How many bits of resolution does the analog to digital converter (ADC) on the Arduino have** 7-bit resolution.

Part C. Resistance & Voltage Varying Sensors

One of the useful aspects of the Arduino is the multitude of analog input pins. We'll explore this more now.

FSR

a. What voltage values do you see from your force sensor?

0 to 1000V.

b. What kind of relationship does the voltage have as a function of the force applied? (e.g., linear?)

Exponential (little force to actuate the sensor but substantial force to push it to the outmost range).

c. In `Examples->Basic->Fading` the RGB LED values range from 0-255. What do you have to do so that you get the full range of output voltages from the RGB LED when using your FSR to change the LED color? Cycle through the FSR values mod256 as follows:

```
if(res_value <= 255){
    setColor(0, res_value%256, 255-res_value%256);
}
else if(res_value < 511){
    setColor(res_value%256, 255-res_value%256, 0);
}
else if(res_value < 767){
    setColor(255-res_value%256, 0, res_value%256);
}
```

Flex Sensor, Photo cell, Softpot

a. What resistance do you need to have in series to get a reasonable range of voltages from each sensor? Flex - 2 10KOhm resistors in series (20KOhms)

b. What kind of relationship does the resistance have as a function of stimulus? (e.g., linear?) Flex - Linear

Control the colors of the LED using the above sensors (including FSR) Flex: `part_c_flex.MOV` FSR: `part_c_fsr.MOV`

Part D. I2C Sensors

Accelerometer

a. Include your accelerometer read-out code in your write-up. See `media/part_d.MOV`.

```
#include <Adafruit_LIS3DH.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
Adafruit_LIS3DH lis = Adafruit_LIS3DH();

void setup(void) {
    // Start Monitor and allocate memory for I2C devices
    Serial.begin(115200);
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println("SSD1306 allocation failed");
        for(;;); // Don't proceed, loop forever
    }
    else{
        Serial.println("OLED initialised");
    }
    Serial.println("LIS3DH test!");

    if (!lis.begin(0x18)) {   // change this to 0x19 for alternative i2c
address
        Serial.println("LIS3DH allocation failed");
        for(;;);
    }
    Serial.println("Accelerometer initialised");
    delay(2000);

    // Report polling rate for Accelerometer
    Serial.print("Range = "); Serial.print(2 << lis.getRange());
    Serial.println("G");
    Serial.print("Data rate set to: ");
    switch (lis.getDataRate()) {
        case LIS3DH_DATARATE_1_HZ: Serial.println("1 Hz"); break;
        case LIS3DH_DATARATE_10_HZ: Serial.println("10 Hz"); break;
        case LIS3DH_DATARATE_25_HZ: Serial.println("25 Hz"); break;
        case LIS3DH_DATARATE_50_HZ: Serial.println("50 Hz"); break;
        case LIS3DH_DATARATE_100_HZ: Serial.println("100 Hz"); break;
        case LIS3DH_DATARATE_200_HZ: Serial.println("200 Hz"); break;
        case LIS3DH_DATARATE_400_HZ: Serial.println("400 Hz"); break;

        case LIS3DH_DATARATE_POWERDOWN: Serial.println("Powered Down");
break;
        case LIS3DH_DATARATE_LOWPOWER_5KHZ: Serial.println("5 Khz Low
Power"); break;
```

```
        case LIS3DH_DATARATE_LOWPOWER_1K6HZ: Serial.println("16 Khz Low
Power"); break;
    }

     // Set Display
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0,10);
    display.println("Accelerometer");
    display.display();
    delay(2000);
    display.clearDisplay();
    display.setCursor(0,10);
    display.print("X:0 Y:0 Z:0");
    display.display();
}

void loop() {
    // Clear Display
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0,0);

    // Read Accelerometer
    lis.read();
    auto x = lis.x;
    auto y = lis.y;
    auto z = lis.z;

    // Print to display
    display.print("X:");
    display.print(x);
    display.setCursor(64, 0);
    display.print("Y:");
    display.print(y);
    display.setCursor(0, 16);
    display.print("Z:");
    display.println(z);
    display.display();

    // Print to Monitor
    Serial.print("X:  "); Serial.print(lis.x);
    Serial.print("  \tY:  "); Serial.print(lis.y);
    Serial.print("  \tZ:  "); Serial.print(lis.z);
    Serial.println();
    delay(200);
}
```

Part E. Logging values to the EEPROM and reading them back

1. Reading and writing values to the Arduino EEPROM

a. Does it matter what actions are assigned to which state? Why? Yes. Assuming the labels of the states remained intact, if you swapped the functionality of the states the program may not work as intended. For example, if you swap the actions of state 0 and state 1, you would never be able to read the data obtained in state 2 as state 1 would clear the memory.

b. Why is the code here all in the setup() functions and not in the loop() functions? Honestly not sure, but best guess is that you only want the code to run once on a state transition.

c. How many byte-sized data samples can you store on the Atmega328? 1024 samples.

d. How would you get analog data from the Arduino analog pins to be byte-sized? How about analog data from the I2C devices? Ints are smaller than bytes so you can store them straight to EEPROM memory. **e. Alternately, how would we store the data if it were bigger than a byte? (hint: take a look at the EEPROMPut example)** Use `EEPROM.put(data)` combined with an increment of `sizeof(data)`.

2. Design your logger

a. Turn in a copy of your final state diagram. FSM for logger

Part G. Create your own data logger!

a. Record and upload a short demo video of your logger in action. See `media/part_g.MOV`.