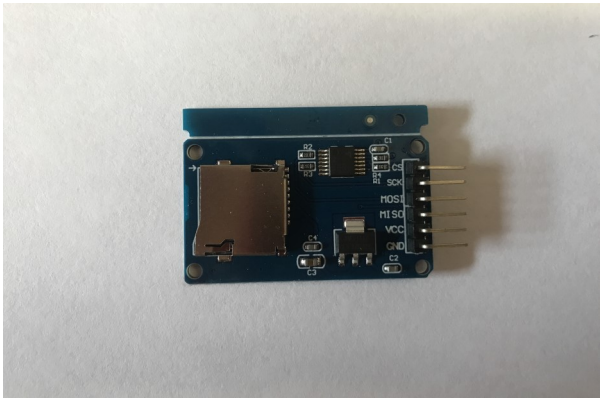
Sensing for Science

Level 4

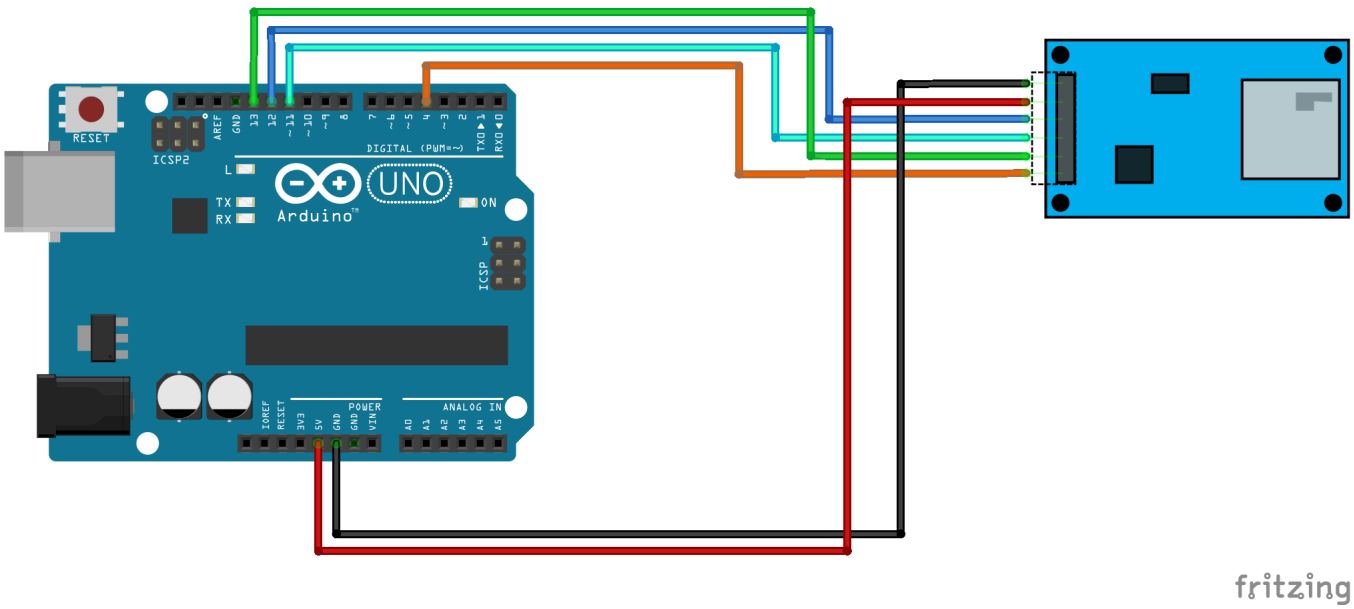
This material is based upon work supported in part by the National Science Foundation EPSCoR Cooperative Agreement OIA-1757351. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Data loggers

We want to go out and collect data from the world. We could take our Arduino circuit and a laptop out to collect data, but that gets challenging with all the equipment and the fear of dropping your laptop. There is the current trend of using IoT (Internet of Things) to send data via the internet, which works great when there is good internet. Out here in Montana, internet connection is not guaranteed. So we will save data locally but using a data logger SD card module. This device will allows us to save data to a micro sd card for us to pull later.



Let's start with testing the module and the SD card. Wire up the following circuit . CS->4, SCK->131, Mosi->11, Miso->12

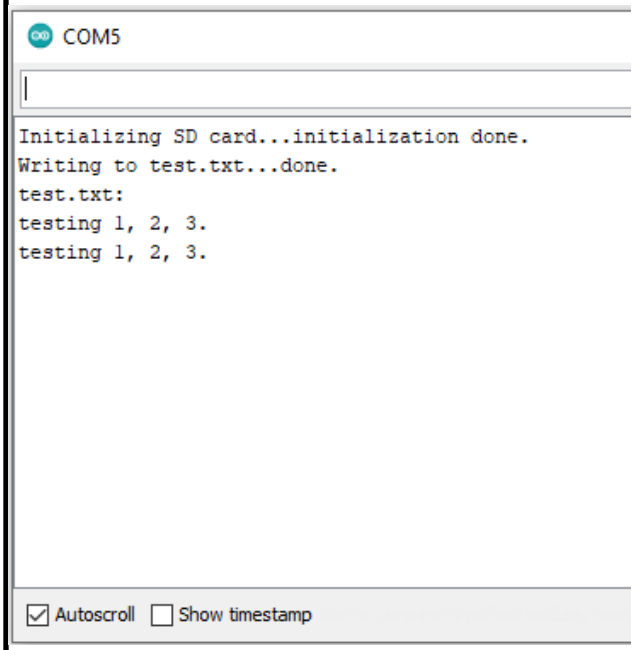


Once you have the SD card module wired up, we can write some code to test the circuit. Copy the code and upload it to the Arduino board. This code is also under examples in the SD card library.

Upload the code to the Arduino board and open the serial monitor. If everything is working correctly, you should see the results shown below. If you get an error message you might have to reformate your SD card (see formatting steps in the instruction video) or your SD card is damaged.

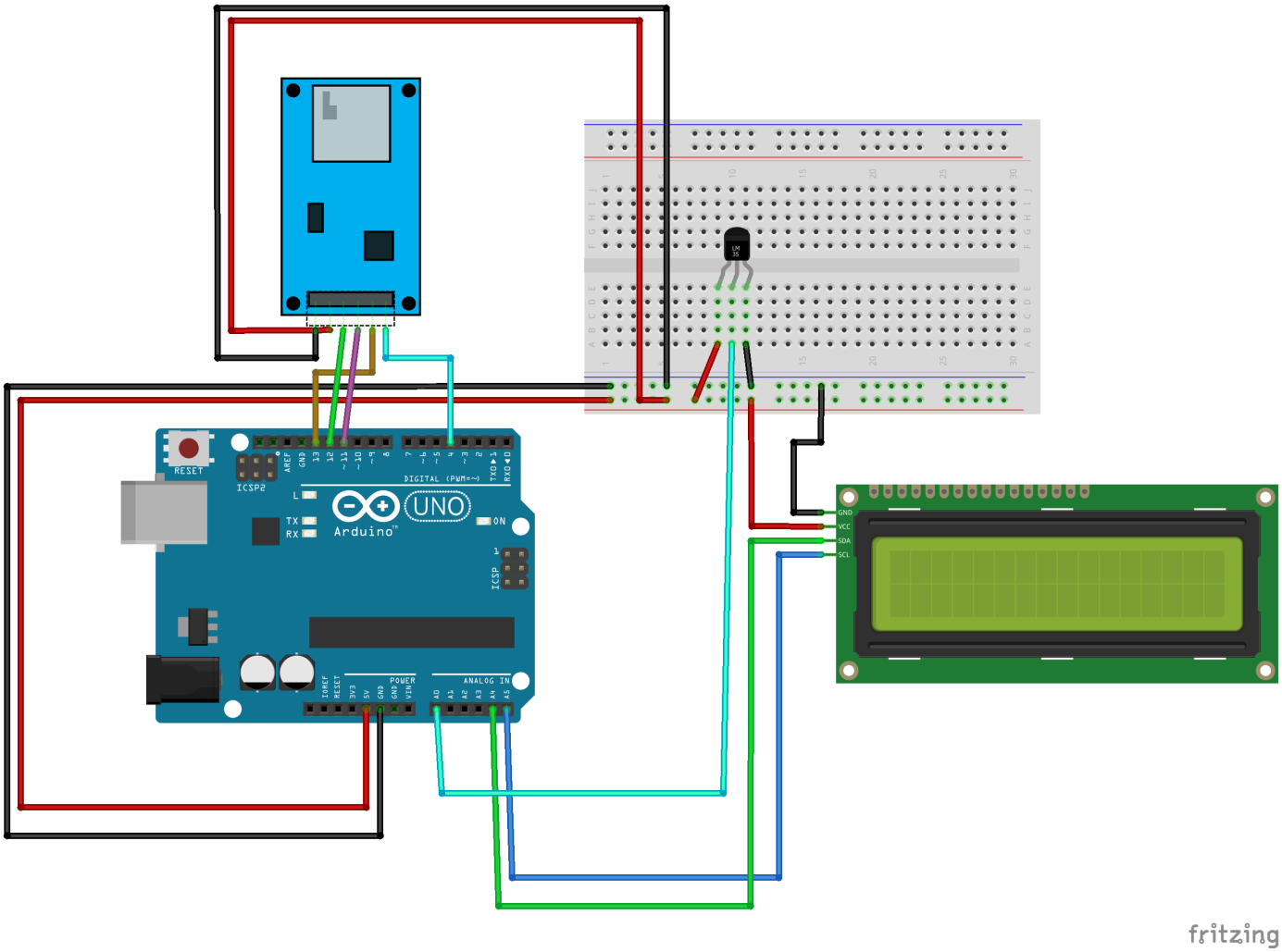
If you have an successful test you should be able to pull the SD card and open the file on the pc. It will should up as a .txt file.

We will break down the code in the next circuit.




```
#include <SPI.h>
#include <SD.h>

File myFile;

void setup() {

  Serial.begin(9600);
  while (!Serial) {
    ;
  }

  Serial.print("Initializing SD card...");

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    while (1);
  }
  Serial.println("initialization done.");


  myFile = SD.open("test.txt", FILE_WRITE);

  if (myFile) {
    Serial.print("Writing to test.txt...");
    myFile.println("testing 1, 2, 3.");

    myFile.close();
    Serial.println("done.");
  } else {

    Serial.println("error opening test.txt");
  }

  myFile = SD.open("test.txt");
  if (myFile) {
    Serial.println("test.txt:");

    while (myFile.available()) {
      Serial.write(myFile.read());
    }

   myFile.close();
  } else {

    Serial.println("error opening test.txt");
  }
}

void loop() {
  // nothing happens after setup
}
```

This next circuit we are going to build will collect temperature data for us at a set interval of time (roughly 10 seconds). This circuit is great when you need to step away but need continuous data collection. You will need a the TMP36, LCD (with I2C), and the data logger. Most of the circuit you will recognize as it is the same temperature sensor from the previous level.



Once you have the circuit built, we can move onto the code. You will notice that the code is starting to get pretty complex. I will break down sections of the code that are new to this project. The code will be on the next 2 pages.

All the libraries we will be using in this code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <SD.h>

LiquidCrystal_I2C lcd(0x27,16,2);

float voltage = 0;
float degreesC = 0;
float degreesF = 0;
const int chipSelect = 4;

void setup() {

  lcd.init();
  lcd.backlight();

  if (!SD.begin(chipSelect)) {
   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("card initialized.");
   delay(5000);
   lcd.clear();
}
```

Setup of the LCD object

Declaring variables for the code

Start up the lcd and turn on the LED back light

This is a check of the SD card. If the card is good and ready, you will get a card initialized message. But if the card is missing or bad you will get a car failed, or not present message and the program stops until the error is fixed.

```
void loop() {

  voltage = analogRead(A0) * 0.004882813;
  degreesC = (voltage - 0.5) * 100.0;
  degreesF = degreesC * (9.0 / 5.0) + 32.0;

  lcd.clear();

  lcd.setCursor(0, 0);
  lcd.print("Degrees C: ");
  lcd.print(degreesC);

  lcd.setCursor(0, 1);
  lcd.print("Degrees F: ");
  lcd.print(degreesF);

  delay(5000);

File dataFile = SD.open("datalog.txt", FILE_WRITE);

 //if the file is available, write to it:
 if (dataFile) {
   dataFile.print("Degrees C: ");
   dataFile.print(degreesC);
   dataFile.print(",");
   dataFile.print("Degrees F: ");
   dataFile.println(degreesF);
   dataFile.close();
lcd.clear();
lcd.setCursor(0,0);
lcd.print("info saved");
delay(5000);

  }
 // if the file isn't open, pop up an error:
  else {
lcd.clear();
lcd.setCursor(0,0);
lcd.print("error SD");
delay(5000);
}
  delay(1000);
}
```

Converting the voltage to useful values

Display the temperature reading

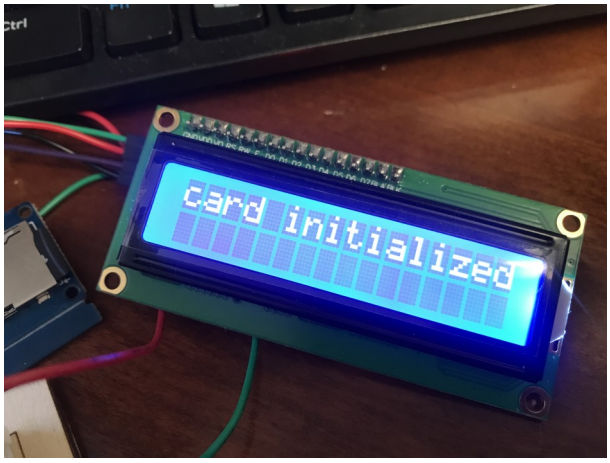
Small delay for read outs

This block of codes is the saving to the SD card. Notice how the print commands work just like the serial port.

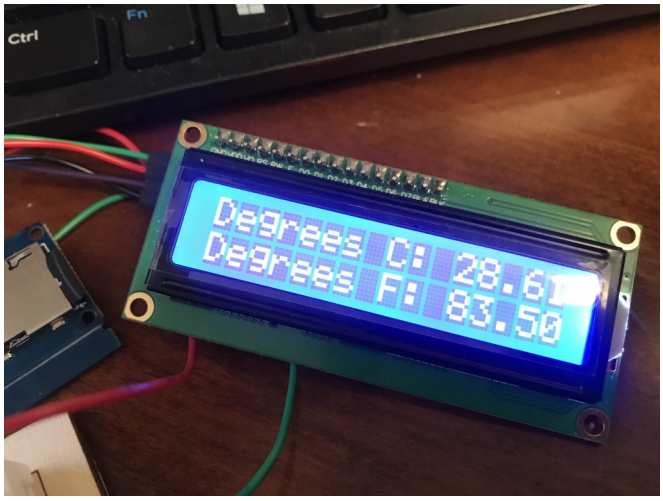
Notification of saved information and delay for 5 seconds. After all the delays should be roughly 10 to 11 seconds before next readying.

If the SD has issues this will give us a error.

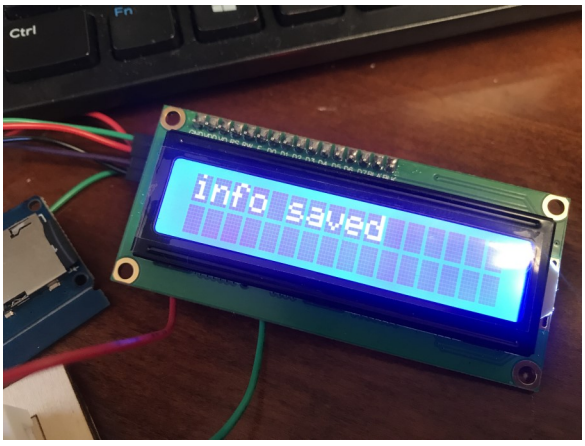
When you have the code ready and uploaded, the circuit will start working right away. You should see the circuit go through its setup code. If everything is ready you should see something like in the image below.



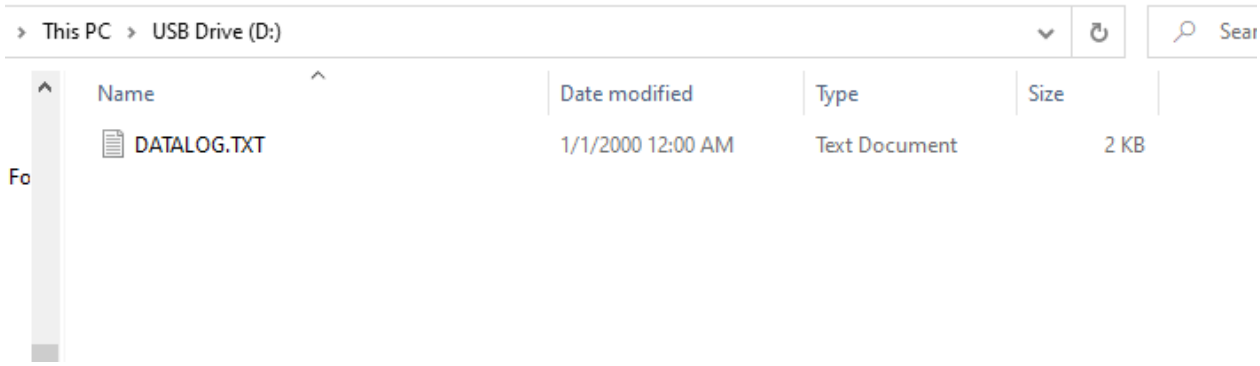
After the setup, the circuit will take a temperature reading and will display on the LCD.



The circuit will save the data to a SD card and you will get a messaging them us that the information was successfully saved.



After we wait for awhile we can turn off the circuit and pull the SD card. Using a SD card we can plug in the pc and open up the drive. You should see a data log file as .txt file. We can open it and see our data.

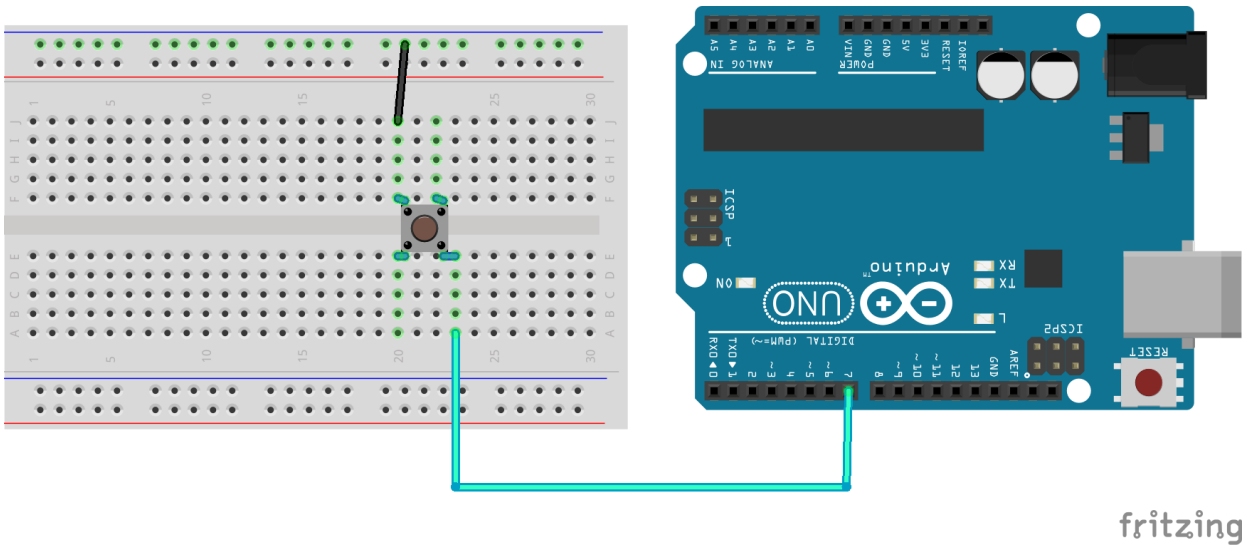


Here is an example of the code in the notepad app. You can import this info into excel to better analyze the date.

```
Degrees C: 32.52,Degrees F: 90.54
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 31.05,Degrees F: 87.90
Degrees C: 31.05,Degrees F: 87.90
Degrees C: 31.05,Degrees F: 87.90
Degrees C: 31.05,Degrees F: 87.90
Degrees C: 30.57,Degrees F: 87.02
Degrees C: 30.57,Degrees F: 87.02
Degrees C: 30.57,Degrees F: 87.02
Degrees C: 30.08,Degrees F: 86.14
Degrees C: 30.08,Degrees F: 86.14
Degrees C: 30.08,Degrees F: 86.14
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 28.61,Degrees F: 83.50
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.10,Degrees F: 84.38
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
Degrees C: 29.59,Degrees F: 85.26
```

What are some down falls of the code we created? How can we improve this code to get us more reliable data?

Now that we have creating a data logger, lets create a circuit that will only collect data when we want it to by pushing a button. We can keep everything from the previous circuit connected. We will just add the button to the design.



This setup will make the Arduino sense a button push. When the button is not being pushed it sits in the "HIGH" status. When pushed current runs through the onboard resistor and sets the status as "LOW". We will use that to trigger the saving of our data. Here is the new code for our circuit.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <SD.h>

LiquidCrystal_I2C lcd(0x27,16,2);

float voltage = 0;
float degreesC = 0;
float degreesF = 0;
const int chipSelect = 4;

int run;
int buttonPin;
```

New variables declared. One for the button and the other for the run command. Run turns orange here because it is used as a keyword in one of the libraries we are using. We are using it as a variable here.

```
void setup() {

 run = 0; //starts stopped
 buttonPin = 7; //whatever pin your button is

  pinMode(buttonPin, INPUT_PULLUP);

  lcd.init();
  lcd.backlight();

  if (!SD.begin(chipSelect)) {
   lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  lcd.clear();
   lcd.setCursor(0,0);
   lcd.print("card initialized.");
   delay(5000);
   lcd.clear();
}


void loop() {

  voltage = analogRead(A0) * 0.004882813;
  degreesC = (voltage - 0.5) * 100.0;
  degreesF = degreesC * (9.0 / 5.0) + 32.0;

  lcd.clear();

  lcd.setCursor(0, 0);
  lcd.print("Degrees C: ");
  lcd.print(degreesC);

  lcd.setCursor(0, 1);
  lcd.print("Degrees F: ");
  lcd.print(degreesF);

  delay(50);
```

Setting values to our variables

Telling the Arduino that we are using pin 7 as an input and also using the onboard resistor too.

```
  if(digitalRead(buttonPin) == LOW)
   {

File dataFile = SD.open("datalog.txt", FILE_WRITE);

 //if the file is available, write to it:
 if (dataFile) {
   dataFile.print("Degrees C: ");
   dataFile.print(degreesC);
   dataFile.print(",");
   dataFile.print("Degrees F: ");
   dataFile.println(degreesF);
   dataFile.close();
lcd.clear();
lcd.setCursor(0,0);
lcd.print("info saved");
delay(5000);

  }
 // if the file isn't open, pop up an error:
  else {
lcd.clear();
lcd.setCursor(0,0);
lcd.print("error SD");
delay(5000);
}}
  delay(1000);
}
```

When the button is pushed, it will run the code to save the data. Otherwise it will continue on looping until we push the button.

After saving data, the loop will continue until pushed again.